NPS52-88-027

NAVAL POSTGRADUATE SCHOOL

Monterey, California

PRELIMINARY WORK ON THE COMMAND AND CONTROL WORKSTATION OF THE FUTURE

Frank E. Harris

John M. Yurchak

Michael J. Zyda

August 1988

Prepared for:

Naval Postgraduate School
Monterey, CA 93943

NAVAL POSTGRADUATE SCHOOL
Monterey, California

Rear Admiral R. C. Austin
Superintendent

Harrison Shull
Provost

This report was prepared in conjunction with research conducted for the Naval Ocean Systems Center and the Naval Underwater Systems Center and funded by the Naval Postgraduate School.

This report was prepared by:

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b RESTRICTIVE MARKINGS
2a SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION / AVAILABILITY OF REPORT Approved for public release; distribution is unlimited.
2b DECLASSIFICATION / DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) NPS52-88-027	5 MONITORING ORGANIZATION REPORT NUMBER(S)
6a NAME OF PERFORMING ORGANIZATION Naval Postgraduate School 6b OFFICE SYMBOL (*If applicable*) 52	7a NAME OF MONITORING ORGANIZATION 1) Naval Ocean Systems Center 2) Naval Underwater Systems Center
6c. ADDRESS (*City, State, and ZIP Code*) Monterey, CA 93943	7b ADDRESS (*City, State, and ZIP Code*) San Diego, CA 92152 Newport, RI 02841
8a. NAME OF FUNDING / SPONSORING ORGANIZATION Naval Postgraduate School 8b OFFICE SYMBOL (*If applicable*)	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER O&MN, Direct Funding
8c. ADDRESS (*City, State, and ZIP Code*) Monterey, CA 93943	10 SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO / TASK NO / WORK UNIT ACCESSION NO

11. TITLE (*Include Security Classification*)
PRELIMINARY WORK ON THE COMMAND AND CONTROL WORKSTATION OF THE FUTURE (U)

12. PERSONAL AUTHOR(S) HARRIS, Frank E., YURCHAK, John M., ZYDA, Michael J.

13a TYPE OF REPORT	13b TIME COVERED	14 DATE OF REPORT (*Year, Month, Day*)	15 PAGE COUNT
Progress	FROM 87/10 TO 88/09	1988 August	147

16. SUPPLEMENTARY NOTATION

17 COSATI CODES			18 SUBJECT TERMS (*Continue on reverse if necessary and identify by block number*)
FIELD	GROUP	SUB-GROUP	Command and control, three-dimensional visual simulation, user interface.

19. ABSTRACT (*Continue on reverse if necessary and identify by block number*)

The modern tactical commander has a flood of sensory and intelligence information at his disposal. A tool is required to sort tht information, allowing the commander to choose the information that is most pertinent to the decisions he must make at that time. This study is the preliminary work on the command and control workstation of the future. The focus of this effort is in two areas. One is a user interface using multiple windows and a mouse controlled cursor. This interface allows the user to set up the display to give him the information he needs in a way that is easy for him to interpret. The second focus is preliminary work on a real-time display that presents the user with a three-dimensional picture of the situation. This initial display uses three resolutions to display large areas of Defense Mapping Agency Digital Terrain Elevation Data with near real time animation.

20 DISTRIBUTION / AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT ☐ DTIC USERS	21 ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL Michael J. Zyda	22b TELEPHONE (*Include Area Code*) (408)646-2305 22c. OFFICE SYMBOL 52Zk

DD FORM 1473, 84 MAR 83 APR edition may be used until exhausted
All other editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE

☆ U.S. Government Printing Office: 1986—606-243

UNCLASSIFIED

Preliminary Work on the Command and Control Workstation of the Future

Frank E. Harris, John M. Yurchak and Michael J. Zyda *
Naval Postgraduate School
Code 52, Dept. of Computer Science,
Monterey, California 93943-5100

ABSTRACT

The modern tactical commander has a flood of sensory and intelligence information at his disposal. A tool is required to sort that information, allowing the commander to choose the information that is most pertinent to the decisions he must make at that time. This study is the preliminary work on the command and control workstation of the future. The focus of this effort is in two areas. One is a user interface using multiple windows and a mouse controlled cursor. This interface allows the user to set up the display to give him the information he needs in a way that is easy for him to interpret. The second focus is preliminary work on a real-time display that presents the user with a three-dimensional picture of the situation. This initial display uses three resolutions to display large areas of Defense Mapping Agency Digital Terrain Elevation Data with near real time animation.

* Contact author.

** This work was supported by the Naval Ocean Systems Center, San Diego, the Naval Underwater Systems Center, Newport and the Naval Postgraduate School's Direct Funding Program.

LIST OF FIGURES

I. OVERVIEW

A. THE COMMAND AND CONTROL WORKSTATION

The amount of information that the modern day commander must assimilate is staggering. There are inputs from a multitude of sensors: NTDS, radars, sonar. There are situation reports, intelligence reports and messages of various types continually arriving. On top of this, the commander has at his disposal a database of intelligence information made up of charts and publications that cannot be quickly accessed. The commander needs a tool that gives him instant access to this information, presenting it in a way that allows him to quickly grasp the facts and make a decision. He should be able to easily manipulate the data to show only the information pertinent for the situation at hand and that data should be up to date and instantly intelligible. It should be easy to display new or amplifying information as situations change and evolve.

This study is the preliminary work for the design and implementation of a tool to provide the commander easy access to command and control information, the Command and Control Workstation of the future (CCWF). The goal of this work is the development of the visualization tools and techniques required to build a prototype workstation with a flexible user-friendly interface and a real-time, three-dimensional display that possesses the characteristics described above. The CCWF is implemented on inexpensive workstations available in the Graphics and Video Laboratory of the Department of Computer Science at the Naval Postgraduate School.

1. Discussion

The graphical and computational power of low cost workstations can be utilized to build a effective system that meets or exceeds our goals for a command and

control workstation. Areas that we have explored are: networking between multiple workstations and mini-computers, the use of multiple windows to help provide an easily managed, user friendly interface, and a real time three-dimensional display.

a. Color

Due to expense, color graphics have not been utilized in the development of current command and control displays. The advent of inexpensive graphics workstations has made the use of color not only possible but a cost effective means to make the display easier for the user to interpret. The user must read and decode the textual information before he can extract the information he requires. With such systems, it is hard to find the particular bit of information needed from a screen full of text. The textual representation can be improved by coding the information. In coding, the original stimulus or information is converted to a new form and displayed symbolically. Coding can make the information easier to find and interpret than direct representation. Current command and control displays use a special military symbology to represent information but do not use color as a coding medium. Color is an effective dimension of coding that has been shown to improve human response time. [Ref. 1] Color is particularly effective in searching tasks where one is scanning through data looking for information of a specific type, such as scanning a command and control display for enemy aircraft. With the dropping cost of color displays, it no longer makes sense to utilize only monochrome when the combination of current symbology with color can increase the effectiveness of our displays. Color will particularly improve performance for the monitor who has other tasks besides watching the scope. The radar operator whose job is tracking contacts on the radar screen knows what each contact is. Color will make it easier for the supervisor to occasionally glance at the screen and quickly comprehend the information presented.

b. Networking

A command and control workstation has to rely heavily on networking. It has to handle several thousand contacts from tactical data systems such as NTDS, as well as entries from various intelligence sources. These contacts have to be continuously updated over the network to insure that the information that is displayed is as up to date as possible.

The network can be used to allow multiple workstations to access a common database of intelligence and amplifying information that allows the user to display digitized intelligence photos or provide parameters on weapon systems and platforms. The displays can obtain and display fixed land based objects of interest like surface to air missile sites or targets for an air strike. This ability makes the workstation useful for planning and training purposes as well as command and control in the heat of battle.

The network also has to handle any communication between the workstations. Along with an exchange of information at the system level, a mail system can help a user more effectively manage his time by evaluating an incoming message and informing the user that mail has arrived by displaying the message's precedence and subject line. The user can then, at his convenience, open a message window and examine his mail.

c. Windows

The user interface is an important design decision in any system, especially when designing a command and control workstation. The purpose of command and control workstations is to assist the user by displaying needed information quickly, in a manner that is easy to understand. The user interface needs to be easy to operate and quick to change, allowing the user to manipulate the display with a minimum

of fuss and bother. A technique that has become common on commercial workstations and personal computers is the use of windows. A window is simply a virtual display. One or more windows can be open and displayed on a computer monitor at one time. Operations that are fairly standard on all window systems allow the window to be moved about the screen and the size changed. The windows can be tiled so all windows can be seen or stacked where one window overlaps all or part of the other windows. Push and pop commands cause a window to be drawn on the bottom or top of the stack. Most window interfaces use a mouse controlled cursor to make these operations even easier but a "track ball" familiar to users of current naval command and control systems can be used with equal ease. Multiple windows allow the user to set up his display depending on the current situation and his particular preferences. This set up can be easily changed as the need of the user changes. A possible window set up for a command and control workstation might be: one small window displaying information such as time, own ship's position, course and speed; two windows with two dimensional radar type displays, one set up with a short range showing surface tracks, the other long range air tracks; another window set up with a chart of the area showing map information as well as contact information; and finally a simulated three- dimensional view that gives the user a "real world" perspective on the situation. By using a mouse, amplifying information can be shown on objects in any of the windows simply by pointing and clicking. As the situation changes, windows can be opened, closed, moved, and the size modified to form a display that presents the required information in a manner that can be easily understood and acted upon.

d. Three-Dimensional View

There is a saying that a picture is better than a thousand words. This is not only true because we can get more than a thousand words of information into one picture

but because a human user can extract the information quicker and easier. We have spent our whole lives evaluating situations in three-dimensions. Every time we drive a car or walk across the street, our mind instantly evaluates what we see and determines the appropriate action. Up to now, that has not been possible in the command and control environment. We have relied on two dimensional displays and written status boards that present a picture of the situation that our mind must interpret before making a decision. If the information can be presented in the three-dimensional form that a user is used to dealing with, he can more intuitively grasp the situation and act accordingly. As usual, the science-fiction writers have lead the way for technology to follow. In the final battle scene of the movie "STAR WARS", the commanders coordinating the attack are all observing and making their decisions based on a large three-dimensional holographic display. Current technology isn't quite to that point yet but we can present excellent, real-time, three-dimensional animation on current graphics workstations.

How useful can such a three-dimensional display really be? The modern day commander has to keep thousands of facts and figures in his head about weapons platforms, sensors, weapons, effective ranges, what is on what ship, what planes have what capabilities, etc. He then has to apply this information to the current situation, picturing in his head the weapons envelopes relative to himself and other contacts. How much easier would it be if he could look at a three-dimensional representation showing contacts and selected weapons envelopes? He could almost instantly assess the situation and act.

An amphibious landing is another instance where this three-dimensional view is valuable. Landing areas and boat lanes can be marked. By using a digital terrain database, the land can be accurately displayed. Landmarks, friendly assets and enemy positions can be displayed. Such a display gives the commander an accurate picture of

the situation to which he can immediately understand and react. Solutions might even seem more obvious when he can actually look the situation over. A three-dimensional display of this nature can give the commander more understandable information in a shorter time then status boards, two-dimensional displays and contour maps ever will.

This three-dimensional display has more uses than just tactical situations. Planning and training are two other areas where this display is useful. To plan an air strike against a land target, we can first enter all intelligence information on radars, weapons installations, airfields, and possible targets. We can then display weapons and radar envelopes and look for the best possible route. After plans are made, the display can be used to brief the air crews who will be flying the mission. They can get an accurate picture of what they can expect to see as the mission proceeds. The simulated display can be supplemented with digitized intelligence photos, if available. The display can then be used to monitor the actual mission in progress.

Three-dimensional displays with real time animation have a place in the command and control environment. Such a display gives the user a more intuitive grasp of the conditions, making it easier and faster for him to interpret the situation. Current low cost graphics workstations are capable of producing a quality three-dimensional display that can greatly aid command and control.

B. SCOPE OF THIS STUDY

The scope of this effort is clearly quite broad. It builds upon previous work bringing us a step closer to the ideal command and control workstation. The focus is in two areas. The first area is to provide a user interface that is consistent with the overall goals for the system. We use an interactive system of multiple display windows controlled with a mouse. The second area is to start development of a realistic, real-time,

three-dimensional display using Defense Mapping Agency Digital Terrain Elevation Data. The goal of this system is to provide the data structures and algorithms required to realistically display the sea/land environment with real time motion simulation.

II. FOCUS

A. PRELIMINARY RESEARCH

Two computer graphics research projects at the Naval Postgraduate School have lead directly to the research on the current command and control workstation [Refs. 2, 3]. Adams developed a two-dimensional command and control display using standard NTDS symbology with color coding added. This system received track information over a local ethernet from another workstation. Adams' work looked at the networking capabilities of graphics workstations as applied to a command and control system and showed that the workstation can be effectively used as a color NTDS display handling the standard operations [Ref. 2]. What remained was to improve the user interface and test the practicality of using windows to provide multiple, separately controlled displays on one workstation.

The second project "FOG-M" started the groundwork for a three-dimensional display, by using graphics workstations to develop a flight simulator using actual digital terrain elevation data [Ref. 3]. The FOG-M system, while producing excellent images, had several limitations which had to be overcome before it could be used in a command and control display. An important factor in the design of a simulator is for it to look realistic but it is often not important for it to be accurate. Graphics "tricks" can be employed to produce a realistic looking image with a minimum of work for the computer. In the FOG-M system, the terrain is only drawn out to a distance of two kilometers from the viewing position, although it gives the appearance of being much farther . For use in a real time command and control system, the terrain must be depicted

accurately and enough of it must be drawn to show the area of actual visibility. At a height of eye of only two meters, visibility to the horizon is three miles [Ref. 4].

B. WINDOWS

There are many windowing systems available on graphics workstations. They are all the same in principle and provide about the same operations but differ in the actual user interface. No industry standard has yet emerged but any system's operation can be simulated by writing an abstract system over the top of the resident system. This conforms to good software engineering practices by making the code more portable since changes to the resident system only effect the procedures that make up the abstract window system. Griggs developed a window manager abstraction that modeled the window systems popular on some personal computers [Ref. 5]. This windowing system can be modified to meet the needs of the command and control workstation by providing multiple window capability to the work done in [Ref. 2]. By adding an extra layer of software, some of the performance of the host workstation is lost. The degradation of the system is not significant in the two-dimensional system but for the three-dimensional display the performance of the host computer needs to be pushed to its limit to provide the best possible animation of complex scenes involving thousands of filled polygons. For this purpose, the native window manager on the IRIS 4D graphics workstation is utilized to provide the user interface.

C. THREE-DIMENSIONAL DISPLAY

It is important that a tactical three-dimensional display render the scene accurately and respond quickly. It is relatively easy to draw a complete and accurate picture when time is no object but a tactical display must be real-time to convey the required information. In a tactical environment, the situation can change in a short period of time. If the information is several minutes old before it is displayed, it loses much of its

usefulness. Enough information must be displayed to depict all aspects of the situation so speed cannot be achieved by leaving out information. The display must update often enough to give the user a sense of movement. This animation provides some of the intuitive information that makes this type of display especially useful.

Three-dimensional scenes are normally rendered using filled polygons to approximate curved surfaces. This cuts down on the computation time required to show a scene but accuracy is lost with the resolution of the polygons. In general, the more polygons you draw the better the scene looks. It also takes a proportionally longer time to draw the scene. The power of a graphics workstation is generally quantized by the number of z-buffered, Gouraud shaded polygons that can be drawn in one second. Since each manufacturer measures this value differently, comparisons are not definitive, although it is a good indication of the capabilities of an individual model. The power of graphics workstations is increasing at a fantastic rate with each new model that comes into production. The graphic workstations in the Naval Postgraduate School's Graphics and Video Laboratory have kept up with these advances, staying on the leading edge of current technology. The IRIS 2400, released in 1985 can draw 650 z-buffered, Gouraud shaded polygons per second. The current models in use, the 3120 and 4D/70G have rates of 1,000 and 5,500 respectively. The next upgrade to the 4D is to be released in May 1988 and is advertised to draw 60,000 z-buffered, Gouraud shaded polygons per second. For a further comparison of these machines, see Table 2.1 [Ref. 6]. With new experimental architectures and multi-processor systems, the future is even brighter. Each new advance makes the rendering of more complex scenes possible and the need for even more powerful machines more apparent. Even with the advances of the near future, machines will be pushed to their limit with real-time three-dimensional displays. Each new advance's additional graphical and computational power is quickly utilized. The key

Table 2.1 COMPARISON OF SILICON GRAPHICS' WORKSTATIONS					
Machine	Z-Buffered, Gouraud shaded Polygons per second	Z-Buffered, Gouraud shaded Polygons per 15th second	CPU MIPS	Flat-Shaded, Non-Z-Buffered Polygons per second	Flat-Shaded, Non-Z-Buffered Polygons per 15th second
IRIS-3120	1,000	67	2	5,000	333
IRIS-4D/70G	5,500	367	10	25,000	1,667
IRIS-4D/70GT	60,000	4,000	13	32,500	2,167
*IRIS-4D/MP?	200,000	13,333	120	300,000	20,000
* under development					

to producing a good, three-dimensional display still depends upon the programmer writing efficient code to draw the best possible picture with the fewest possible polygons.

The FOG-M and VEH visual simulators, developed at NPS, used Defense Mapping Agency digital terrain elevation data from Fort Hunter Liggett California for their displays [Refs. 3, 7]. There are graphic techniques or "tricks" that can be used to make a scene look realistic with fewer polygons. Some of these tricks were used in FOG-M and VEH to draw exceptional scenes with a minimum number of polygons. Some of these techniques sacrifice accuracy for performance. Unlike a simulation, a tactical three-dimensional display must be both realistic and accurate. New methods must be found to reduce the number of drawn polygons without giving up the accuracy. Images in FOG-M and VEH were only drawn to a maximum distance of two kilometers reducing the number of polygons that must be drawn. The appearance of greater distance was achieved by using perspective settings like different lenses on a camera. Setting a wide field of view in the perspective call gives the same effect as a wide-angle lens. Objects that are in the background appear much farther away. Since the vertical resolution is not affected by this technique, an additional scaling of the actual elevations gives a realistic, although not always accurate, display of terrain. Since a tactical display must also be accurate, all the terrain that can be seen must be drawn in the correct perspective and the elevations must be accurately represented. It requires a tremendous number of polygons

to represent the visible terrain if all the terrain is drawn at the same resolution. Since things in the distance cannot be seen as clearly as close by things, they do not need to be drawn at the same resolution as those that are closer. The number of polygons drawn to represent the terrain can be reduced if more polygons are used to represent the terrain closest to the viewpoint while the terrain in the distance is drawn at a lower resolution using fewer polygons.

D. SUMMATION

This study brings the command and control workstation of the future a couple of steps closer. Both user interfaces with windows and techniques for drawing terrain in a three-dimensional display have been studied and implemented. The focuses are to provide a user-friendly program framework for future work on the command and control workstation and to make an accurate and realistic real-time three-dimensional display of digital terrain elevation data.

III. WINDOWS AS A USER INTERFACE

The command and control workstation of the future (CCWF) is a tool designed to present tactical information in a way that the user can easily interpret. Like any tool, its only as good as its user interface. The CCWF must be designed with the user in mind. A device that is complicated, or hard to use is likely to go unused even if it is functionally perfect. Windows have been used to improve the interface of personal computers and commercial workstations. Their use can also enhance the user interface of the command and control systems used by the Department of Defense.

A. WINDOW DISPLAYS

Windows are the basis of the user interface for the CCWF. They provide flexibility and the effect of many displays for the price of one. The technique of multiple windows as a user interface is still maturing in the computer industry. There are many diverse window management systems on the market but no industry standard has yet emerged. The services provided by all the systems are similar and a viable user interface can be developed under any of these systems.

1. Multiple Windows

The CCWF uses multiple windows on a single screen. Each window is used to provide a particular type of information in a format that is easy for the user to understand. There are windows for two-dimensional radar display, chart display, three-dimensional display, and auxiliary information. All these windows cannot be shown on a screen at one time. The user can select which windows to show, where on the screen it is displayed and how big it is. The windows can interactively be opened, closed, moved, or

resized. They can be pushed or popped in order to be drawn under or over the other windows. These operations allow the user to change the display according to the particular needs of the current situation. The window system must know where the window is to be drawn, what size to make it, what to draw in the window, and how it should be drawn, relative to the other windows.

2. Mice and Menus

The standard computer interface is the typewriter-type keyboard. Even with special function keys, the user has to remember special codes and it takes multiple keystrokes to accomplish simple operations. With a mouse or track ball controlled cursor, an operation can be executed simply by pointing the cursor at a selection displayed in a menu on the screen and hitting a button. The menu presents choices in easy to understand text. All the user has to remember is how to get the menu and how to press a button.

The mouse controlled cursor can also select certain objects from a window. For example, the user can bring up amplifying information about a contact by pointing at the contact on the NTDS screen and pushing a button. This technique is currently used on standard NTDS displays. The CCWF expands on this idea by having different cursor modes. The user changes the cursor by pointing at a menu in the display. Each cursor has a separate function relative to the window it is in.

3. Mice in Multiple Windows

Using a mouse with multiple, overlapping windows is a much more complex problem then with a single window or tiled window structure. The system must determine what window the mouse event occurs in. It would be nice for this operation to be transparent to the user. The user points and the system knows which is the referred to window. On a multi process system, the window system must also be able to determine with which process the user is interacting.

B. WINDOWS ON THE IRIS

The IRIS graphics workstations from Silicon Graphics Inc. were selected as the primary graphics device for the NPS Graphics and Video Laboratory. These are inexpensive, high power, high resolution, color computing systems for two-dimensional and three-dimensional computer graphics. The IRIS provides a powerful set of graphics primitives and custom VLSI circuits combined with conventional hardware and software [Ref. 8]. The heart of the system is a special VLSI chip called the Geometry Engine. A set of these chips make up a geometry pipeline that accepts points, vectors, polygons, characters, and curves in user defined coordinates and transforms them to the screen coordinate system. The pipeline does all rotations, transformations, clipping, and scaling to draw the objects the correct size at the correct location. The IRIS has the capability to use multiple user interface devices. Besides the standard keyboard, the unit has a mouse with three control buttons. Optional equipment includes button and dial boxes. A window management system is included as standard software on the IRIS.

1. Multiple Exposure Window Management System

The Multiple Exposure Window Management System (MEX) is a user interface environment that controls multiple windows [Ref. 9]. This system is a standard part of the operating system on the IRIS 4D and an optional program that can be run on the earlier models. It provides multiple windows, mouse controls and a pop-up menu system. Multiple windows can be controlled by one process and several processes can have windows open on the display at one time. This is a very powerful system that can be used to design an exceptional user interface.

a. The Event Queue

The user communicates with the computer system through input devices. He pushes a function key, types a message on the keyboard, uses a mouse, etc. There are

two ways for the input device to communicate with a process. The first is for the process to read the value of every device. This is done every time through the main program loop, even though the value of the device has not changed. This is known as polling the devices. Another method is to send the controlling process a message whenever a change in a device has occurred. The message is placed on a queue and the controlling process checks to see if any messages have arrived. Polling a device is less efficient when values are not frequently changing. Why read the device and act on the input when no change has occurred? With an event queue, the system places a message on a queue each time a change in state occurs on selected devices. The process can check for any and all input by simply using a while loop to read and act upon all the messages in the queue.

```
while ( queue not empty ) do
   {
   process message
   }
```

This way the process only takes time to act on changes and not on checking all the devices every cycle through the loop.

This study is concerned with communication to a single process but is implemented on a multi-process system. The window manager on IRIS is designed to work in a multi-process environment where several processes running concurrently might have displays open on the screen. One process must be designated to receive input. This is called input-focus on the IRIS system and can be changed interactively using the mouse and MEX menus. All events are placed on the queue of the process with input-focus assigned.

b. Pop-Up Menus

Menus are provided by the IRIS window manager to make the interface between the user and the system as easy as possible. These menus are controlled by the

mouse and appear at the cursor location when the right mouse button is pressed [Ref. 9]. Keeping the right mouse button pressed and moving the mouse highlights the different menu selections. When the correct menu selection is highlighted, the mouse button is released. The selection is then executed and the menu disappears.

The MEX system uses a predefined pop-up menu to control window operations. Figure 3.1 shows the operations that can be executed from this menu. This menu is activated by placing the cursor in the title box of the window that requires the action and holding down the right mouse button.

There are also user defined menus. These menus are defined in the application program. When this menu is executed, a message is placed on the applications event queue. Input from the mouse and menu goes to the process with input focus.

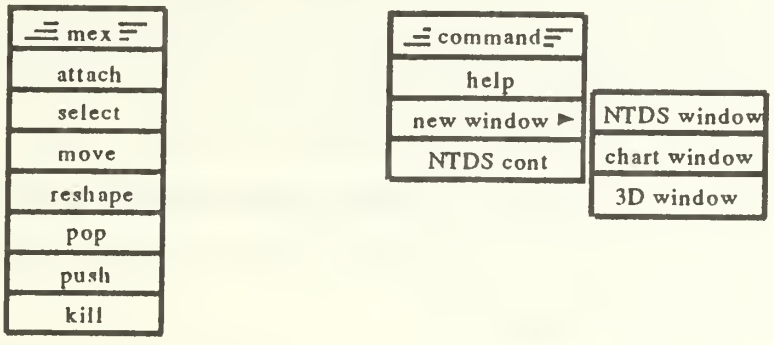


Figure 3.1 MEX System Menu MEX User Defined Menu

c. Window Management

The IRIS window manager has an excellent set of routines to manage windows from within a user application. The application program can open and close windows, changing the display based on the program or user input. For example, if the application receives a message from another unit, it can open a window and display a message telling the user what has happened. The user can then close the window with some predefined command, mouse button, return key, etc.

Whenever a window is opened, the system returns an identifier to that window. The graphics window identifier (GID) is a long integer that is used to identify what window is being referred to in a program. This value is used to close a window or set the window to receive graphic input. Only one window at a time can be set to receive graphics input. Most of the MEX procedures act on the window that is set to receive graphics input.

The system allows the programmer to set certain constraints on a window. Maximum and minimum size can be specified, the X to Y aspect ratio can be set, or the window position and size can be fixed. All these commands can be changed by the program after the window has been opened. If constraints are set, the user cannot violate these constraints from the MEX interactive menu.

All the operations in the MEX command menu can be executed from within the program, allowing the programmer to devise a window control system that meets the needs of a particular application.

C. IMPLEMENTATION

MEX was designed to run on a multi-processing system and works best with many processes, each with one graphics window. This study is primarily looking at a system

where one program, the command and control workstation, is controlling multiple windows. The built in functions of MEX need to be expanded to keep track of the different windows.

Enhancing the program manager to meet the needs of the command and control workstation can be done in two ways. An abstract window manager can be constructed using the software engineering principle of information hiding. This abstract window manager would consist of procedures that would call the appropriate MEX functions and keep needed information in its own data structures. The programmer calls procedures in the abstract manager and does not deal with MEX directly. A basic abstract manager was developed by Larry Griggs in [Ref. 5]. The other method would be to develop a data structure that would maintain the information needed to control the windows using MEX calls directly. This method does not make the software easily portable, but it is more efficient. By removing a layer of procedure calls between the user application and the system window manager, performance is improved and the complexity of the program, as a whole, is improved.

The MEX pop-up menu system is not always the best way to present choices to the user. If there are many possible choices, pop-up menus can become confusing and hard to manage. To keep the interface simple, multiple menus are used with each menu designed to present its choices in a way that makes it easy for the user to make his selections.

1. Menus in a Simple Interface

Operation of the command and control workstation allows the user to make many choices about the configuration and operation of the workstation. Placing all the choices in one MEX menu with roll-over submenus presents the user with a menu that is too complex to be easily used. Another problem is with the MEX system. There is no

way to select several things from a menu at once. There are two parts in the solution to these problems. The first is to modify the menu depending on the possible choices the user can make from his current state and which window the cursor is over at the time of the call. This allows the user to see only the choices that make sense for his current state and window. The second part of this solution is to take infrequent choices, such as configuration and set-up options and options that don't work well in the pop-up menu system, and place them in other menu systems that present the choices in a better way.

a. Multiple Menus

Presenting different menus for different situations and windows requires that the program be able to tell which window the cursor is over when the user calls a menu. This is not a built in function of MEX but is fairly easy to do. The windows are drawn in a stack. When a new window is opened, it is placed on the top of the stack. When a push is ordered for a particular window, it is moved to the bottom of the stack where it is drawn under all the other windows. A pop operation moves the window to the top of the stack. The main program needs to keep track of where each window is on the stack. When a menu is called, the program starts at the top of the stack and checks each window to see if the coordinates of the cursor position falls inside the window boundary. The first window that succeeds is the window the user is referring to in his request. The following code shows a C function that returns the GID of the window where the cursor is or a -1 for error.

```
/* array of graphic window identifiers */
/* windows[0] is the top window */
GID windows[10];

/* function which_window returns the GID of the top
widow where the input X/Y parameters lie inside
the window boundary */

     GID which_window(mousex,mousey)
     {
```

```
        found = FALSE;   /* initialize */
        i = 0;

        while((not found)or(i <= number of windows))
        {
          /* make window to be checked current graphics window
          so following procedures will refer to correct window */
          winset(windows[i]);

          /* get window coordinates */
          getsize(sizex,sizey);
          getorigin(originx,originy);

          if (mousex > originx) and (mousex < originx + sizex) and
             (mousey > originy) and (mousey < originy + sizey) then
             found = TRUE;
          else
             i = i + 1;
         }/* while */
         if (i > number of windows)
            return(-1); /* not in any of the windows */
         else
            return(windows[i]); /* the GID of the first window found
*/
}
```

b. A Virtual Control Panel

A typical NTDS display has the capability to filter out contact types. This simplifies the display by only showing the contacts that the user wants to see. This feature is controlled by a panel of hardware switches on the console. The command and control workstation of the future can show multiple NTDS displays on one screen. The controls for multiple displays are too varied and complex for a hardware control panel. Interactive menus must be used to control the different features of the workstation. A virtual control panel is used to set up an NTDS display window. The user positions the cursor over the NTDS window he wishes to change. The pop-up menu for this window has NTDS control as one of the selections. When this selection is executed, a new window is opened as a virtual control panel (see Figure 3.2). The cursor is bound to the inside of this window and no other action can be executed until the user is through setting the controls. The virtual control panel is a table of boxes, each box is labeled according

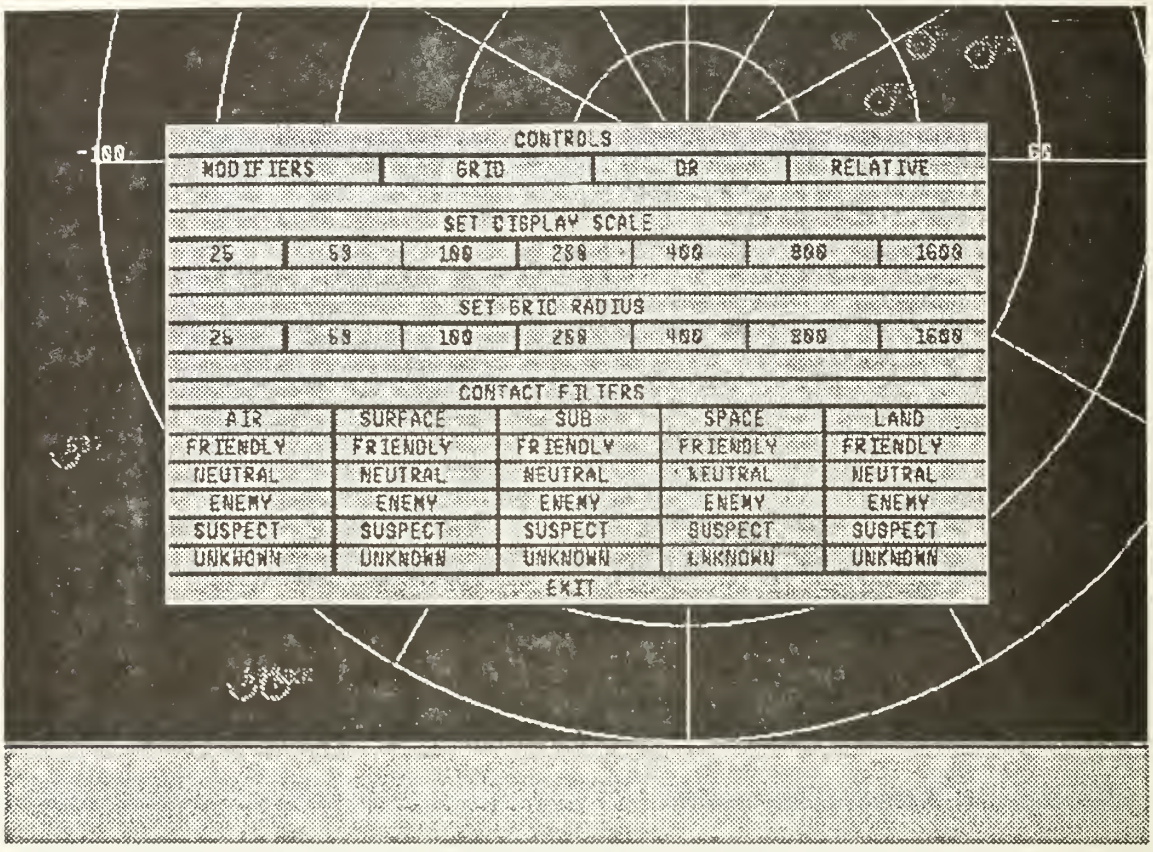


Figure 3.2 The NTDS Virtual Control Panel

to its function and color coded to indicate its setting. Red is off and blue is on. The "switch" is toggled by pointing the cursor at the appropriate box and clicking the left mouse button. The color changes indicating a new setting for that switch. Changes to the panel can be made until the exit box is pressed. The window is then closed and any changes are entered into the user record of the window where the execution of the pop-up menu occurred. This allows each of the NTDS displays to be controlled separately and display a variety of information. One window can show the air picture, one the surface and another can be set to display only hostile units.

2. An Abstract Window Manager

The purpose of this preliminary version of the command and control workstation is to demonstrate the functionality of using multiple windows and menus in developing the user interface. The starting point of this program is the commander's display system developed by Rodney Adams [Ref. 2]. The commander's display system uses color graphics to display a single NTDS radar display. Windows are not used in this program. A simple abstraction of a window management system was developed by Larry Griggs for his network monitor [Ref. 5]. Griggs' window manager simulates the user interface used in some popular personal computers. The windows are controlled by clicking the mouse in certain predefined regions in each window. For example, holding the mouse button down in the lower right corner of the window then moving the mouse to a new position, changes the size and shape of the window. This is in contrast to the MEX system, where windows are controlled with a special pop-up menu. Many of the procedures used in this abstraction are direct calls to MEX procedures.

The current version of the CCWF uses parts of the NTDS display from Adams, and an expanded and modified version of the Griggs window manager. This program runs on the IRIS 3120 under the MEX window management system. This preliminary program demonstrates the operations required by the user interface for the command and control workstation.

a. Multiple Windows

The abstract window manager solves some of the problems in the MEX system. It keeps track of each window, its size, position, and content. It has procedures that evaluate in which window a mouse event occurred. Additions have been made to allow constraints to be placed on each individual window. Predefined constraints can be set by the programmer for maximum size, minimum size, and X/Y aspect ratio. The

aspect ratio is used for the NTDS display to keep the scale the same in the x and y direction. This keeps the perspective constant so the display looks "right" when the size of the window is changed. The shape of the window remains constant. The information to be displayed in the window is scaled to match the window size. Multiple open windows can be shown tiled, Figure 3.3, or overlapped, Figure 3.4, and moved around to give the user a picture that is easiest for him to interpret.

(1) Data Structures. Window information needs to be saved when multiple windows are used in one program. The abstract window manager hides this information in a package of procedures that can be called by the programmer to control the multiple windows. The following is the data structure used in this abstraction.

```
/* structure used to record information on each window */
typedef struct WindowType
{
/* the user provides these fields when he opens a window */

   char     title[50];      /* title to be displayed in title bar */
   Object   *obj;           /* ptr to the object to be drawn in window */
   short    BackgroundFlag;/* boolean: does window always stay in
                              backgnd? if so, it can't be moved,
                              popped or resized and should be as
                              large as the screen to avoid losing
                              a normal window beneath it */
   long   refCon1,refCon2; /* user uses these for any purpose he
                              likes. Refcon one is used as pointer
                              to user structure */

/* MIWM keeps these fields current */

   int    wid;             /* IRIS-mex-provided unique window id */
   long   wx,wy;           /* x,y dimensions of the window */
   long   orgx,orgy;       /* screen coords of lower left corner of window*/
   short  max,min,aspect;  /* are constraints set */
   long   maxx,maxy;       /* constraint values */
   long   minx,miny;
   float  aspratio;
} Window;
```

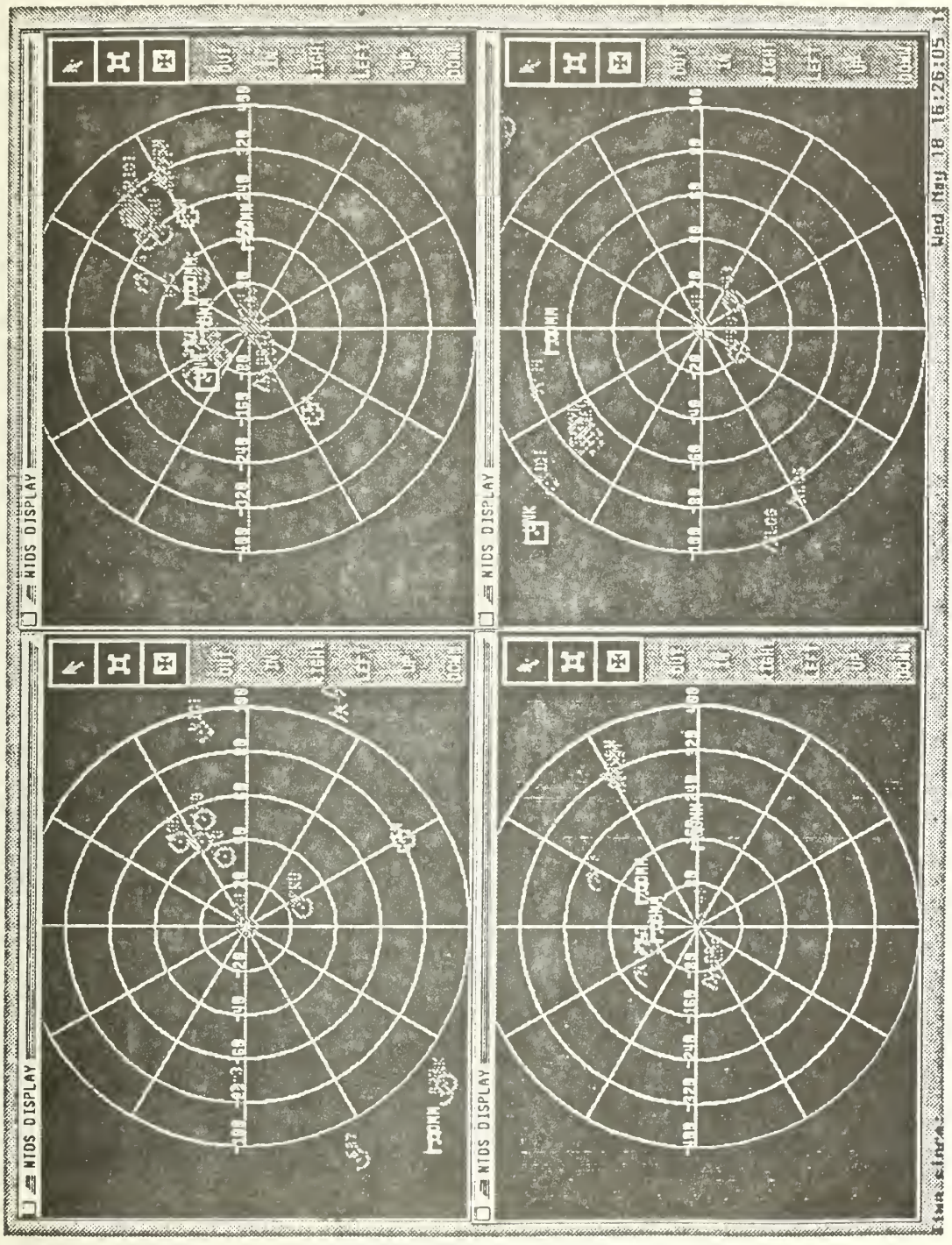


Figure 3.3 Tiled Windows

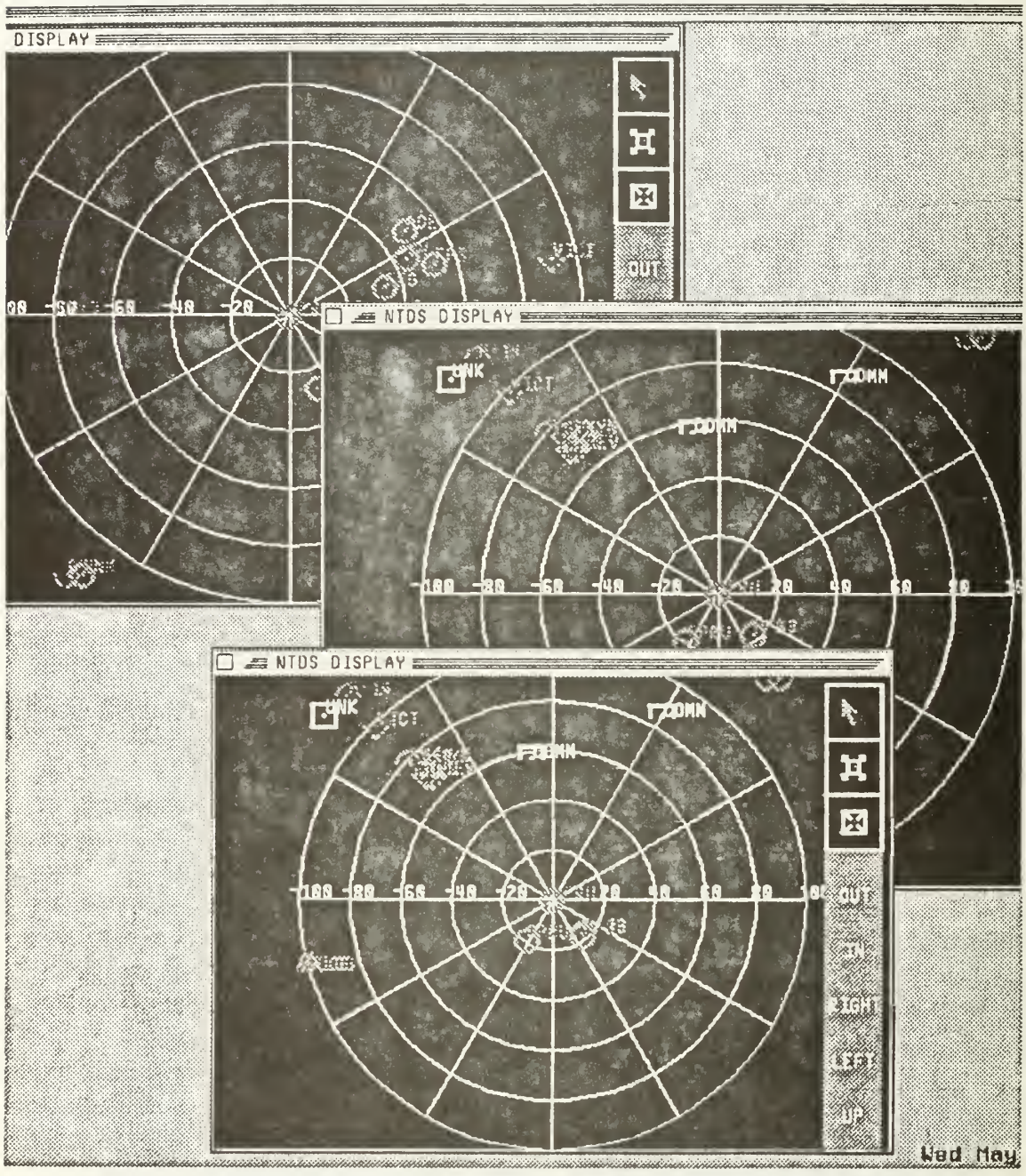


Figure 3.4 Layered Windows

This record contains information that can only be accessed by the procedures in the window management package. When the process needs to know what window a mouse event occurred in, a package procedure called whichwindow() returns the information.

Another requirement imposed by multiple windows is keeping track of what the user wants in each window. This information is application dependent and should not be part of the abstract window manager. Each window has a user record that has information about what the user wants drawn in each window. The following structure is the user record format used in the CCWF implementation.

```
/* Data structure used to share user interaction
   information, one user record is assigned to each
   window. A record for an open window will be marked
   as used */
typedef struct {
    Object user_object;          /*the object for this window*/
    int    win_type;             /*int: type of window*/
    Window *wind;                /*pointer to window record*/
    short  used;                 /*boolean: is this window in use*/
    int    center;               /*index of contact at grid center*/
    short info_box[MAXSYMBOLS]; /*which contacts show info box*/
    short grid;                  /*boolean: show grid or not*/
    short modifiers;             /*boolean: display or not*/
    short dr;                    /*display dr updates*/
    short relative;              /*relative or true display*/
    short zoom;                  /*number to indicate zoom status*/
    short updown;                /*number to indicate shift left right*/
    short leftright;             /*number to indicate shift up down*/
    short cursor;                /*user can select 6 different cursors */
    float grid_radius;           /*user selected NTDS grid size*/
    float scale;                 /*"zoom" is used to alter this value*/
    float x_offset;              /*user-selected display offset*/
    float y_offset;              /*user-selected display offset*/
    short Fair;                  /*friendly air*/
    short Nair;                  /*neutral air */
    short Uair;                  /*unknown air */
    short Sair;                  /*suspect air */

        More Boolean Flags to filter contact types
      one entry for each possible contact designation

}user_struct;
```

These records are modified by the user through the use of various menus. There is a maximum effective number of windows that can be open at one time. The data structure

for the CCWF implementation is arbitrarily set at ten windows. The number of windows that can be effectively handled depends on the complexity of the window contents. More then four NTDS windows slows the system to an unacceptable level on the IRIS-3120.

In order to know which user record to change, the window system must know what window the cursor is over and what user record is associated with that window. Pointers bind the window structure and the user structure together when a new window is opened.

b. The Windows

The current system uses five types of windows. A window is used to display the NTDS control panel, another is used to display an intelligence photograph of a selected contact type. These two windows are only open for a short time and must be closed before work continues. There is a background window that cannot be closed or sized. This window just provides a background for the display. It can be used to display help information or the three-dimensional view can be implemented in this window. The two remaining are window types which show information, a 2-D radar and a chart window. Other types of information windows can be easily added to the system. Multiple numbers of each of these windows can be used. The functionality of this system can be demonstrated with only one type of information window, the NTDS window. A chart window selection is presented in the menu but its selection opens an empty window.

(1) Selecting Items From A Window. The IRIS graphics library routines allow the user to select objects from the screen. This is controlled with the cursor and the left mouse button. When the left mouse button is pressed the system is put in "pick" mode. Pick mode allows the program to determine what designated objects are within a defined region around the cursor's location. This process is used by the abstract

window manager to determine if the user is trying to move or reshape the window. It is also used to pick certain objects from the contents of the window.

(2) NTDS Windows. The NTDS window shows features that can be used in any of the windows. Besides the virtual control panel, there are three different cursor modes, and 6 control boxes that act as a permanent menu. The different cursor modes are selected by picking the cursor from the permanent menu on the right side of the window (see Figure 3.5). Any of the cursors can be used to manipulate the windows

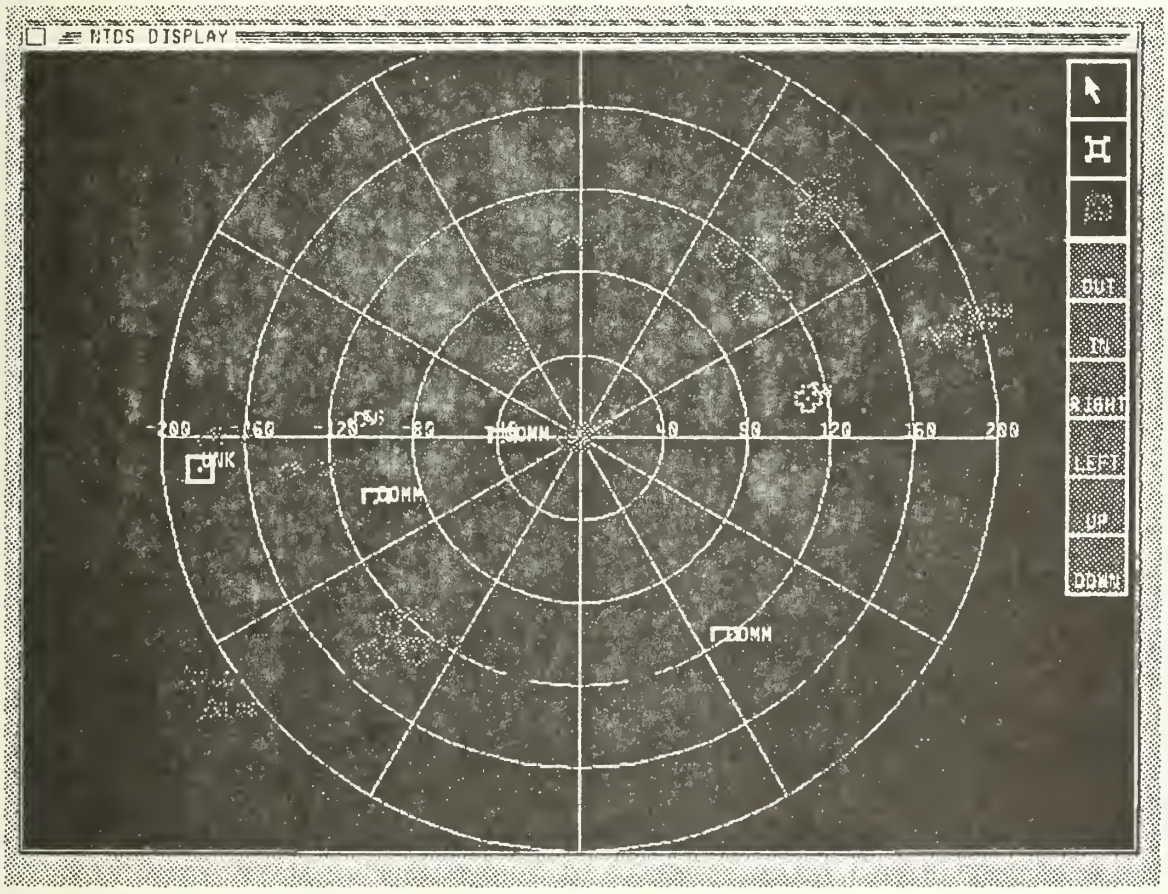


Figure 3.5 NTDS Window

or select items from the permanent menu. The top cursor, an arrow, is the default cursor. When the arrow cursor is active and a contact symbol is picked from the window, an information box shows the contact's course, speed, etc. The middle cursor, the one that looks like a picture frame, displays a picture of the contact. This is just an example. This cursor could be used to show any amplifying information. For example, it could display a menu and allow the user to select what type of amplifying information he wants to see. The menu choices would vary depending on the information available about that particular contact type. The third cursor looks like a sight and changes the window origin. The center of the grid becomes the contact selected by the user. All movement is

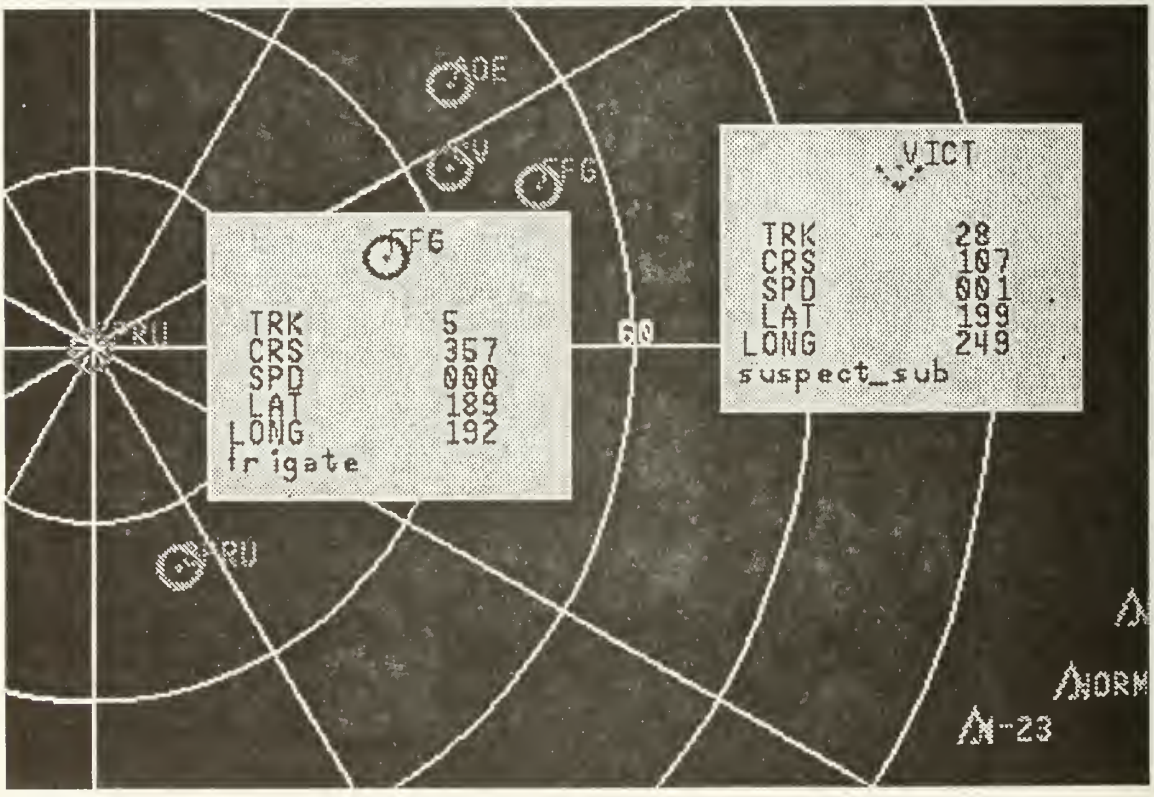


Figure 3.6 Information Box

displayed relative to this contact's course and speed. The remainder of the permanent menu allows the display to be shifted in the window to show the part of the area that is of interest. Figure 3.7 shows an NTDS window that has been shifted up and to the right.

There are also zoom-in and zoom-out controls that focus on the center of the display. The techniques used in the NTDS windows can be used in other windows to give the user quick and easy control over the display.

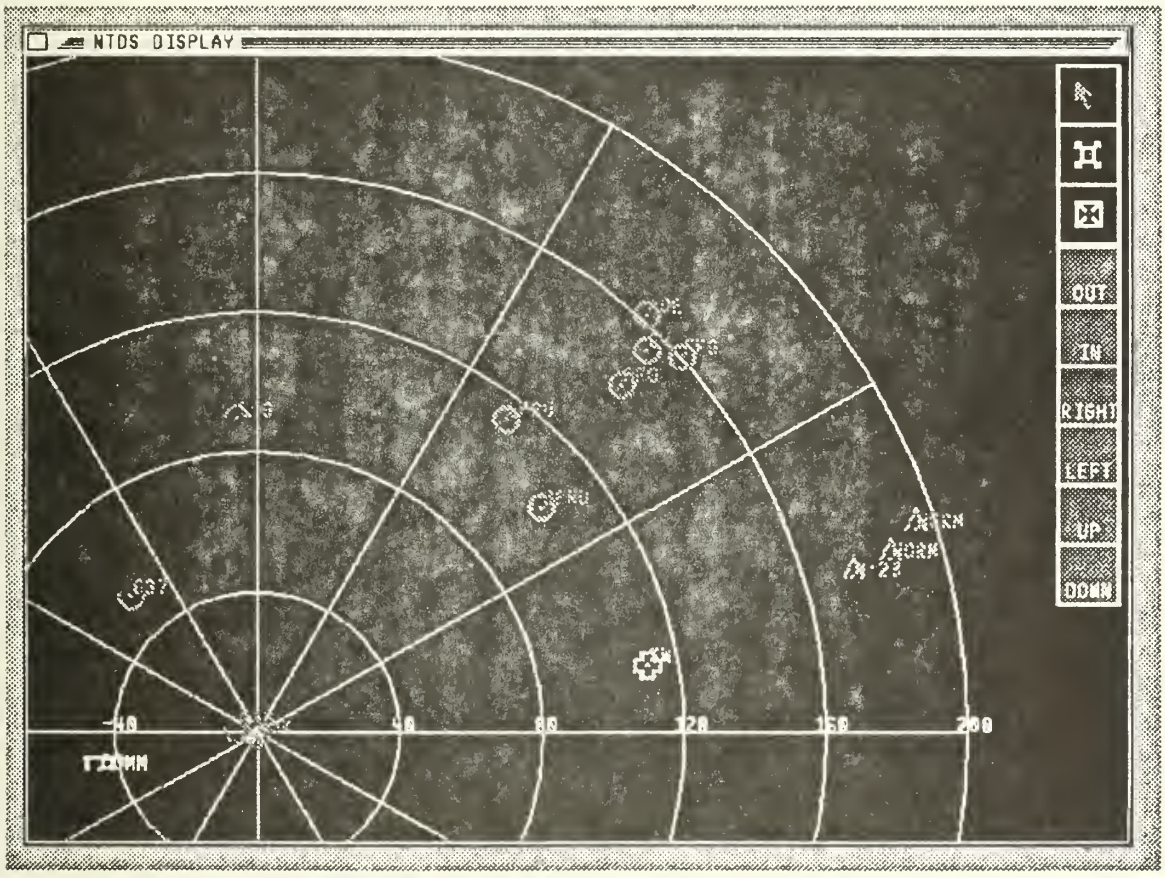


Figure 3.7 Shifted Grid Display

3. An Alternative to the Abstract Window Manager

Window controls are application dependent and can not be easily generalized into an abstract manager. The application and the host computer define what information needs to be saved about the windows. The two data structures used in the abstract manager are bound together by pointers. The complexity of the program is improved if the structures are combined and an application specific window manager is used. A structure used in the prototype of the three-dimensional display places information required by all windows in one record type.

```
typedef struct
{
/* the user provides these fields when he opens a window */

char  title[50];          /* title to be displayed in title bar */
int   Wintype;            /* int representing window type, NTDS, Chart,
                             3D, etc. Will determine what type of
                             information is to be drawn in the window.*/
short BackgroundFlag;     /* boolean: does window always stay
                             in backgnd? if so, it can't be moved,
                             popped or resized and should be as
                             large as the screen to avoid losing a
                             normal window beneath it */
int   Gid;                /* IRIS-mex-provided unique window id */
long  pointer;            /* pointer to window specific record,
                             record will contain information
                             that is specific to the window type */

/* these values can also be obtained from MEX calls */

long  wx,wy;              /* x,y dimensions of the window */
long  orgx,orgy;          /* screen coords of lower left corner of window*/

/* individual constraints for each window */

short max,min,aspect;     /* are constraints set */
long  maxx,maxy;          /* constraint values */
long  minx,miny;
float aspratio;
} Window;
```

Information that is window specific is placed in another structure that is pointed to by a field in the main window record. The following record is the window type specific structure used for the three-dimensional display.

```
typedef struct
{
   int    lat;          /* lower left whole latitude of pos */
   int    longg;        /* lower left whole longitude of pos */
   float  posx;         /* position relative to lat longg in yards*/
   float  posz;         /* position relative to lat longg in yards*/
   float  posy;         /* altitude in yards above sea level */
   float  crs;          /* motion of viewpoint */
   float  spd;          /* motion of viewpoint */
   float  lookdir;      /* Chg view direction without course chg */
   float  lookang;      /* angle down from vis horizon */
   float  max_vis;      /* set maximum range, current 60 NM */
   int    attached;     /* contact number attached to 0 if free */
   short  gridlines;    /* show grid lines ? */
}viewpoint;
```

The MEX system controls windows with a pop-up menu that is called from the window title bar. The window manager abstraction used in this study controls the windows with the cursor in certain defined, but unlabeled, regions of the window. Both techniques work and provide an easy to use interface. If an application specific window manager needs to be implemented on top of the native window manager, performance and complexity are improved if the applications window manager is as close as possible to the native window manager. All the MEX system functions except push and pop can be used directly from the MEX menu. There is no way to keep the window stack straight if the system executes pushes and pops directly. The MEX system menu cannot be changed by the application programmer to remove the unwanted functions. An application menu system can be developed that calls the MEX routines from the application program and stores the needed window information. Menu controlled windows are easy to use and provides more flexibility then the method used in the Griggs abstract manager.

D. CONCLUSIONS

Multiple windows with mouse controlled menus form a very powerful and flexible user interface for the CCWF implementation. There is no real industry standard for

window management systems. Each manufacturer has designed their window interface to work efficiently with specific hardware and applications in mind. Until a standard interface emerges, different techniques should be considered. Design decisions should not be based on the system used by some subset of the window oriented computers. By programming a window manager that is application specific and that uses the native window manager, a user interface that meets the needs of the program can be developed on any of the window systems. A production model of the CCWF will be a special purpose computer. An efficient window manager for a special purpose computer must be designed to take advantage of the hardware and provide all the functions required by the specific application. The MEX system is an excellent window manager and will work very well in this application. The new release of the IRIS window manager in the IRIS 4D/GT places a token on the event queue whenever a window is pushed or poped. A function is also provided that will indicate where a particular window is in the stack. The CCWF, implimented on the IRIS 4D/GT, can use the native window manager directly. The only information the application program needs to save is the GID of each window and the information about what should be drawn in each window.

IV. DIGITAL TERRAIN DATA

To model the terrain over a particular area of the earth's surface, a database representing the actual terrain is required. This database should consist of a set of points covering the area to be modeled and show the actual elevation of the terrain at these points. In addition it would be nice to know more about the terrain at each of these points so that future applications can add man-made structures, vegetation or color to make the images more accurate and informative. The data used in this study is from the Defense Mapping Agency's Digital Terrain Elevation Data (DTED).

A. DEFENSE MAPPING AGENCY

The Defense Mapping Agency (DMA) was established in 1972, combining the mapping, charting, and geodosy functions of the defense community into one organization. Its mission is to provide Department of Defense users with "complete, credible and effective mapping, charting, and geodetic products, services, and training" [Ref. 10]. The DMA Combat Support Center in Brookmont, Maryland is primarily responsible for the distribution of DMA products to the military services. The product of primary interest for use in a three-dimensional display is the Digital Landmass System (DLMS) Database maintained by the DMA Aerospace Center, St. Louis, Missouri.

1. Digital Landmass System Database

The DLMS database is in two parts, cultural features and terrain. The terrain files have elevation data in meters above mean sea level. The cultural files have information about the terrain. These two files are compatible. The horizontal position of

the cultural data matches the terrain data insuring that features in the cultural files coincide with the proper terrain elevation locations.

a. Resolution of Data

The DLMS database is available in two resolutions, level 1 is intended for broad worldwide coverage and level 2 is intended to cover small areas of interest. The interval between data points for level 1 resolution is three seconds of latitude arc or approximately 100 yards. Level 2 data is presented in finer detail with a data interval of one second of latitude arc. The corresponding interval of longitude changes with increasing latitude. See Table 4.1 for exact intervals [Ref. 11].

b. Cultural Data

Cultural data is a generalized description and portrayal of planimetric features. In other words, what the terrain looks like. This file is a complex data structure that uses digital codes to describe a feature or area. Special codes are used to represent the predominant surface material of a feature or area. A few of the categories used are stone/brick, water, rock, soil, marsh, asphalt, and trees. In addition to the general surface makeup, unique significant features are also stored. These features are divided into three categories: areal features, liniear features and point features. Point features are unique features less then 150m by 150m in size for level 1 resolution and 30m by 30m in level 2

Table 2. Terrain Data Interval

Zone	Latitude	Level 1 lat. long.	Level 2 lat. long.
I	0^o–50^o N-S	3 x 3 seconds	1 x 1 second
II	50^o–70^o N-S	3 x 6 seconds	1 x 2 second
II	70^o–75^o N-S	3 x 9 seconds	1 x 3 second
IV	75^o–80^o N-S	3 x 12 seconds	1 x 4 second
V	80^o–90^o N-S	3 x 18 seconds	1 x 6 second

NOTE: All values in seconds are in terms of arc measure.

resolution. Objects such as isolated structures, radar reflectors, tall buildings, aids to navigation, etc. fall in this category. Linear features are less then 150m in width for level 1 resolution and less then 30m for level 2. This category includes features such as canals, streams, rivers, roads, railroad tracks, walls, fences, etc. Areal features are too large to be considered in either of the other two categories. Included in this category are parking lots, squares, mud flats, storage areas, etc. The cultural data file is very complex, containing much information not touched on here. The data obtained from this file can render a very accurate picture of the chosen area.

c. Terrain Elevation Data

Elevation data is stored in separate files from the cutural features file. Both level 1 and level 2 data are stored as 16 bit integers representing the elevation in meters above mean sea level at each data interval. Level 1 terrain elevation data is the information used in this study to develop a three-dimensional display. The content and specifications for level 1 terrain elevation data are covered later in this chapter.

B. DIGITAL TERRAIN ELEVATION DATA

A command and control workstation for use in a tactical, sea/land environment must have a wide area of coverage. The area of interest for a commander can easily be a circle a 500 miles in radius from his position. The three-dimensional display for this workstation must be equally far reaching. The digital terrain elevation data used in the FOG-M and VEH visual simulations is not sufficient for this application for two reasons. The data does not cover a large enough area. The Fort Hunter Liggett database is an area 35 km x 36 km. This is not sufficient to show the feasability of managing and displaying the quantity of terrain data required for a tactical display. The second reason is that the Fort Hunter Liggett area is uninteresting for a sea/land environment. There is only a small stretch of straight coastline, containing no bays, inlets or islands. The southern

islands of Japan are an area with both interesting terrain and interesting coastlines and islands. It is that DMA data set on which this study relies.

1. The NPS Japan Database

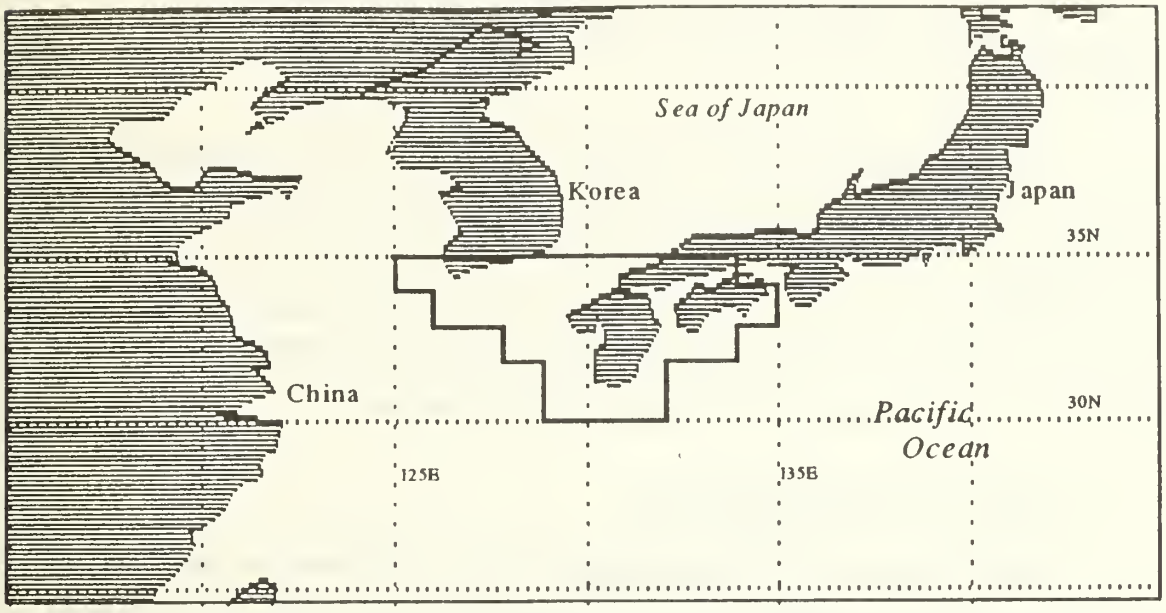
A 10,800 square mile database of level 1 digital terrain elevation data was obtained from the Defense Mapping Agency. The area of interest is the southern part of Japan. This area has many islands and interesting coastal features which make it ideal for testing a three-dimensional display of a land/sea environment. See Figure 4.1 for a map of the covered area.

2. Format

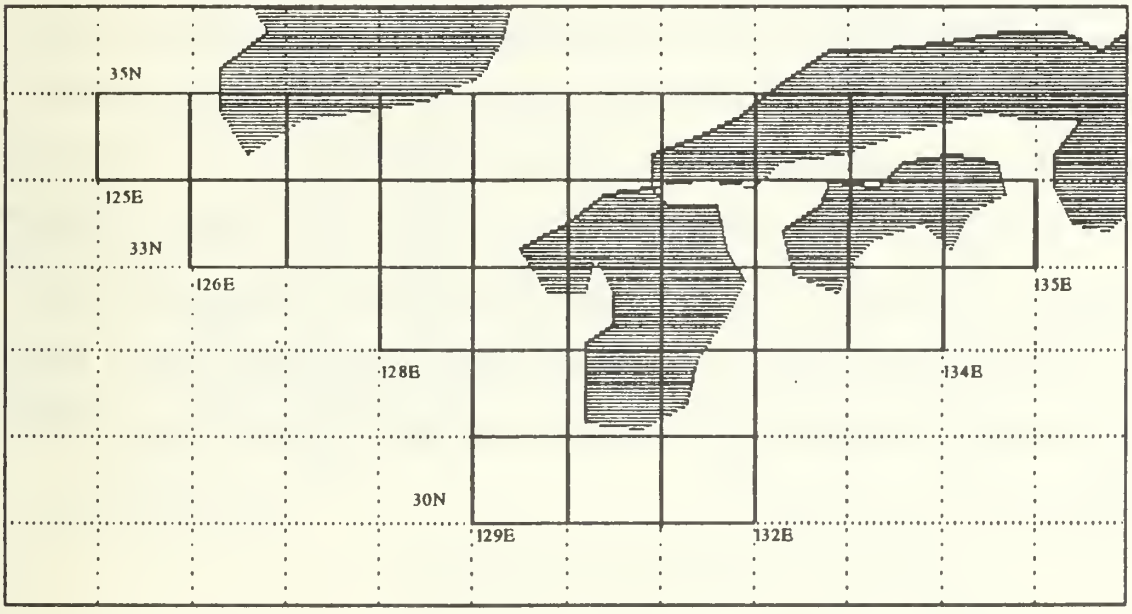
World coverage of DMA level 1 terrain elevation data is divided into blocks, or cells, of one degree latitude by one degree longitude [Ref. 11]. The Japan database used in this study consists of 30 of these cells, distributed on four half inch magnetic tapes. Appendix A contains a list of the latitudes and longitudes of the individual cells on each tape. A map showing the position and coverage of these cells is in Figure 4.1. Each tape contains multiple cells of information. The cells are identified by the latitude and longitude of the southwest corner of the cell (cell origin). The data for each cell consists of three files, a header file, a data file, and a trailer file. The format of the header and trailer files is very simple and uninteresting. Complete format information for the DMA Digital Terrain Elevation Data is presented in Appendix B. The data file consists of three record types:

* 1 Data Set Identification Record (DSI)
  This record contains 648 bytes of information about the data classification, resolution, position and orientation.

* 1 Accuracy Description Record (ACC)
  This record contains 2700 bytes of information about the horizontal and vertical accuracy of the data. The cell can be divided into one to nine subregions. Accuracy data is presented for the overall cell as well as each subregion individually.



Database Region



Database Cells

Figure 4.1 Maps of Japan Database

* Data Record
  There is one record for each line of data points of equal longitude. These records contain the actual elevation data for the cell. A full cell of level one terrain data has 1201 data records. One for every three seconds of longitude.

The area covered by each cell provides an overlap of points having positions of whole degrees of latitude or longitude. For example the elevation data for the point 31N 131E is included in four cells. It is the southwest corner of cell 31N 131E, the southwest corner of 31N 130E, the northeast corner of 30N 131E and the northwest corner of 30N 130E. The content and format of these records is covered below.

3. Reading the DMA Standard Tape

Digital data from the Defense Mapping Agency is delivered on large magnetic tapes. It takes 90 megabytes of disk storage to hold the files for the 30 cells of data obtained for this project. The only machines with enough permanent storage to keep this data are the ISI workstations used in the Multiple Backend Database project at NPS. Each of these machines has a 500 megabyte disk drive. A 125 meg partition on ISIV1 was set aside to store and manipulate the Japan terrain database.

The network of ISI computers has a tape drive capable of reading the DMA standard tapes. *Readone.c*[1] is a simple program that reads one file at a time from the tape. The output filename is passed into the program as a command line argument. Each successive call to the program reads the next file on the tape. The program is written in C and runs on the ISI workstations.

---

[1] Appendix C

4. The Elevation Data

Each cell of level 1 terrain data has sample intervals of three seconds of latitude arc by three seconds of longitude arc for longitudes between 0 and 50 degrees north and south. These points are divided into data records of equal longitude. To provide an overlap, cells points of whole degree latitude and longitude are shared by adjoining cells. A full data record consists of 1201 data values stored sequentially by ascending latitude. 1201 of these records make up the cell. They are stored sequentially by longitude starting with the point of origin (see Figure 4.2).

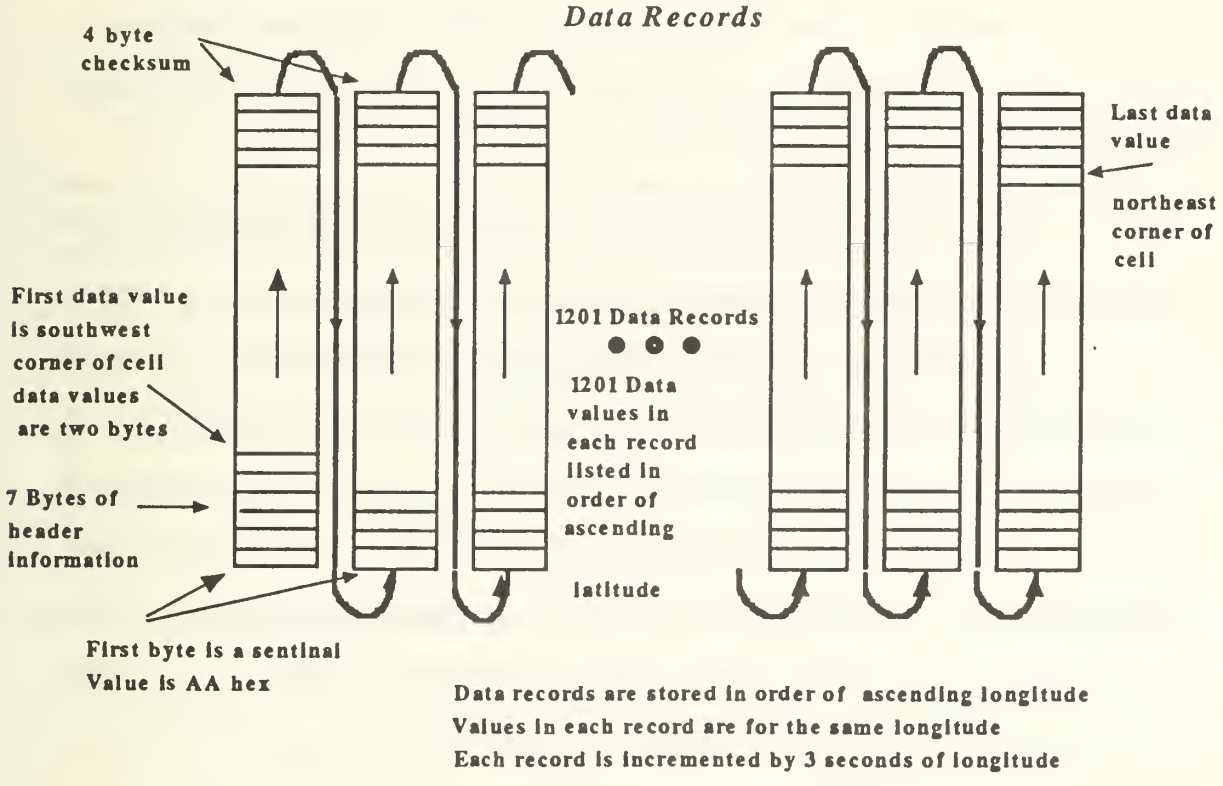


Figure 4.2 Diagram of Cell Data Storage

The Data Record has eight bytes of header information before the data and a four byte checksum after the last data point. The header information consists of:

* 1 byte — recognition sentinel, AA hex
* 3 byte — Sequential count of the block within the file
* 2 byte — Longitude count, a sequential count of the number of data records from the origin (southwest corner of the cell).
* 2 byte — Latitude count, the starting latitude of the record in units of the data interval. Normally zero for a full cell of level one data.

The data values are 16 bit signed magnitude binary integers representing the elevation in meters above mean sea level. The sign is the high order bit. Negative values are not complemented. This gives a value range of +32767 to -32767. The actual range of values is +9000 to -12000 meters. The checksum value is the algebraic sum of all the eight bit values in the record.

The data is easier to use and store in the form of a two-dimensional array of data points without the extra information. *Extract.c*[2] is a program that takes a data file from one cell and writes only the elevation data to an output file for later use. The data is written sequentially, starting with the first data record from the file. This allows it to be easily read into a C array. The filename for the output file is the latitude and longitude of the southwest corner of the data. For example, 31N131E is the file name for the cell with southwest corner at 31 degrees north latitude 131 degrees east longitude. This information is extracted from the cell's data set identification record. The complete format information from [Ref. 12] is provided in Appendix B.

---

[2]Appendix C

V. A THREE-DIMENSIONAL TERRAIN DISPLAY

A real-time, three-dimensional, animated display provides more information and is easier to interpret then a traditional two-dimensional presentation. It is also much more difficult to draw. A scene that shows enough information to be useful is complex. It requires many polygons to accurately depict even simple situations accurately. The command and control workstation is interested in the sea/land environment. The first step to providing an accurate and useful three-dimensional display is to draw land and ocean scenes in a near real-time display.

A. PROBLEMS

Real-time animation is considered to be 30 frames per second. At this rate, the human eye cannot detect the change of frames and the animation appears smooth. It would be nice for a three-dimensional, tactical display to update that fast, but it is not necessary. An update rate of two or three frames per second looks jumpy, but it conveys a sense of motion to the viewer and shows the appropriate, up to date information. A very large area is seen from an altitude of 300 ft. Assuming a field of view of 45 degrees, visibility of 20 miles and DMA level one digitel terrain elevation data, this area has 67,916 data points. It requires 135,832 filled triangles to represent the terrain and takes the IRIS 4D/G about 90 seconds to draw using z-buffering. This is improved upon by drawing the land with filled polygons and drawing the ocean as an underlying blue plane. It requires 54 seconds to draw one frame. Computer upgrades will improve these times but not enough to give an acceptable animation rate. A new method must be found to draw the terrain faster without a loss in accuracy.

Each cell[3] of terrain elevations has 2.8 megabytes of data. Assuming a maximum visibility of 60 miles, four cells of data is the maximum that is required for any one scene. (see Figure 5.1). Four frames contain 11.2 megabytes of data. This data must be available to the program for each frame that is drawn. The digital terrain database being used in this project consists of many islands and a lot of water. The ocean is represented as zero elevation. Space complexity can be improved by compressing the data, i.e. not storing all the zero data.

The third problem addressed in this chapter is actually a set of problems dealing with drawing the terrain. The terrain data is a set of discreet points 100 yards apart. The ocean is represented by points of zero elevation while land is any elevation

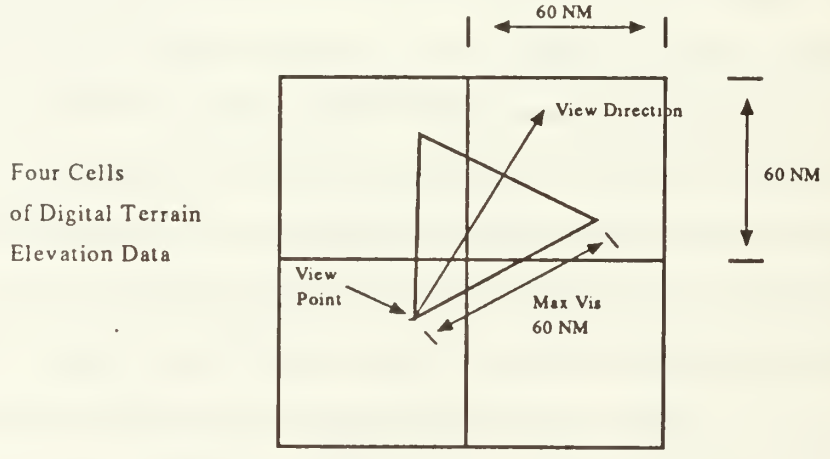


Figure 5.1 View Area Covers a Maximum of Four Cells of Data

[3]Represents one degree latitude by one degree longitude area (Chapter 4)

greater then zero. How is the shoreline drawn between these points? Other problems that are covered are concerned with drawing in z-buffer mode and how to transition between different drawing resolutions.

B. DRAWING THE TERRAIN

The terrain in a tactical three-dimensional display must be drawn in correct proportion to the actual view. Not only must the vertical and horizontal scale model the real terrain but the area displayed must match the area that can actually be seen.

1. Scale

The data interval for digital terrain data is in seconds of arc latitude and longitude. One minute of arc latitude varies between 6,046 ft and 6,108 ft. This corresponds nicely to the use of nautical miles (NM) as the basis for a scale. One NM is 1852 meters or 6076.11 feet [Ref. 11]. A general rule of thumb used in navigation is one nautical mile equals 2,000 yards. The longitudinal scale is dependent on the latitude. At the equator, the distance between whole degrees of latitude and longitude are both equal to about 60 NM. As latitude increases, the interval for degrees of latitude stays about the same but the arcs of longitude converge. At 30 degrees N, the start of our data, the distance between adjacent arcs of longitude is 52 NM. The data point interval for the DMA data base[4] is three seconds of latitude and three seconds of longitude. This equates to approximately 100 yds between points north or south and 86 yards between points east or west. For simplicity, the initial implementation of the three-dimensional display uses the value of 100 yds as the interval between all data points.

---

[4]The database of Southern Japan used in this study (Chapter 4)

The first chapter of [Ref. 11] states that "Relief information to DMA standard digital format is on a three seconds of latitude (approximately 100) meters matrix". Using 1852 meters as a close approximation to one degree of latitude, the actual value is 92 meters for three seconds of latitude. A much better approximation is 100 yards for three seconds of latitude. The actual value is 101.25 yards.

The DMA digital terrain elevation data values are in meters above mean sea level. To keep the scale constant, this value is converted into yards. The relationship, one yard is equal to .9144 meters is used as the conversion factor. This correction is applied when the data is preprocessed into a form that is compatible with the data structure used to store the data.

2. Visibility

Visibility is determined by the curvature of the earth. At an eye height of 6 ft, visibility is three miles. This distance increases with the height of eye of the observer. The area that can be seen is approximated by drawing the area from the view point to the horizon. Since items with an altitude greater then zero can be seen past the visible horizon, a minimum visibility of 10 NM is used to insure that land and objects relatively close are seen. Objects further away that can be seen over the horizon are indistinct and generally are not of interest. A maximum visibility of 60 nm is assigned to simplify determining what data cells are needed. The following formula, derived from, [Ref. 4] is used to calculate the visibility (VIS) in yards given the height of eye (HE) in yards.

$$Vis = 3962.8 \times \sqrt{He}$$

The visibility is used to calculate the area that can be seen in one scene or frame, the visibility triangle (see Figure 5.2). This triangular area is computed using the view point, the direction of view from the view point, the visibility, and the field of view.

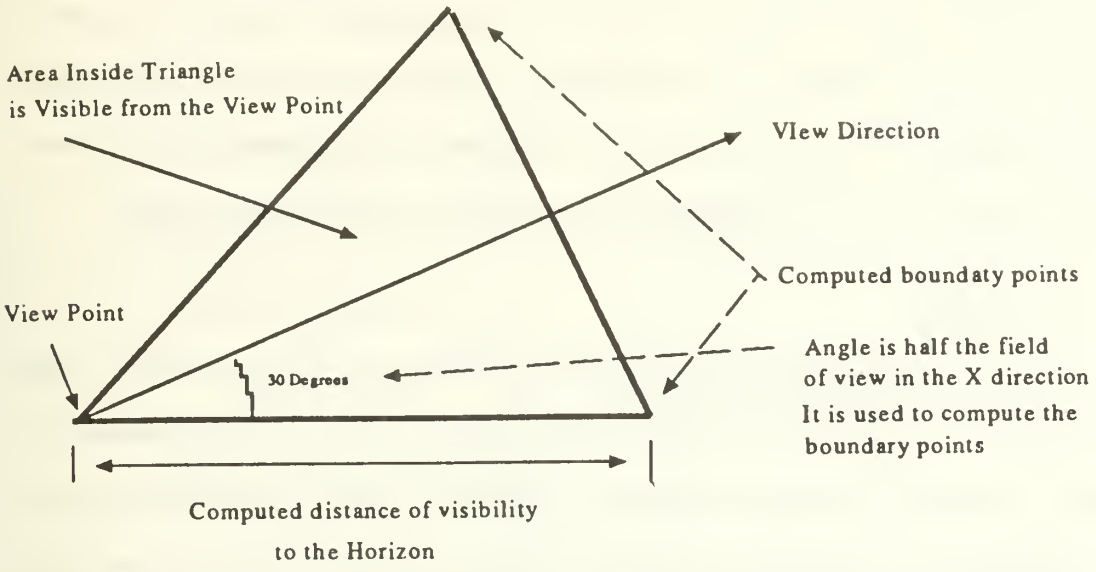


Figure 5.2 Visibility Triangle

3. The Ocean

The ocean can be approximated as a flat plane and drawn as one filled polygon at zero elevation. An area with data values of zero elevation is not drawn as many separate polygons. Instead, a blue base is drawn at zero elevation and only polygons with elevations greater then zero are drawn overlaying the blue base. This method allows scenes with a large proportion of water to be drawn much more quickly. This technique speeds up the the drawing process but causes a problem with the z-buffering. This problem and and the techniques used to correct it are discussed later in the z-buffering section of the chapter.

A sense of movement is detected over terrain by the changing relief of the ground. This can be enhanced by artificially checkerboarding the terrain with two colors. The ocean generally has no relief and can be drawn as a single blue polygon. A sense of motion cannot be derived from a checkerboard pattern since a fill pattern is generally

drawn in screen coordinates vice world coordinates. Such a fill pattern does not show the effects of the view point moving in the world coordinate system. To provide a sense of movement over the water, a set of grid-lines is drawn on the ocean approximating latitude and longitude lines on a chart. The grid-lines are drawn dependent on the altitude of the view point. The ocean and grid-lines are drawn in procedure *draw_terrain*[5].

4. Shoreline

The data can be viewed as a two-dimensional matrix of elevation data. These data values are the elevation of discrete points and do not show what conditions are between the points. The terrain is drawn by assuming that the area between the points can be represented as small planar triangles[6] with the data points as vertices (see Figure 5.3). Since a blue base is drawn first at zero elevation, triangles that represent ocean are not drawn. The land is drawn in alternating colors of green to give a checkerboard effect. Knowing nothing about the transition between points, it is hard to decide where to draw ocean and where to draw land when the adjacent points indicate there is a transition somewhere between them. The resulting picture must show a smooth transition that represents the actual shoreline. The database is converted into polygons by taking four adjacent data points that form a square and drawing it as two adjoining triangles. When all four data points are either zero or greater then zero both triangles are drawn the same, as either both ocean or both land. The problem is when the four points are mixed. One solution that works nicely is to draw a transition or shore when there is only one of the four points that is greater then zero. If two, three or all four of the points are greater then both triangles are drawn as land. This leaves four different ways to draw one triangle

---

[5]Appendix E

[6]Triangles are used to insure that the resulting polygon is always convex.

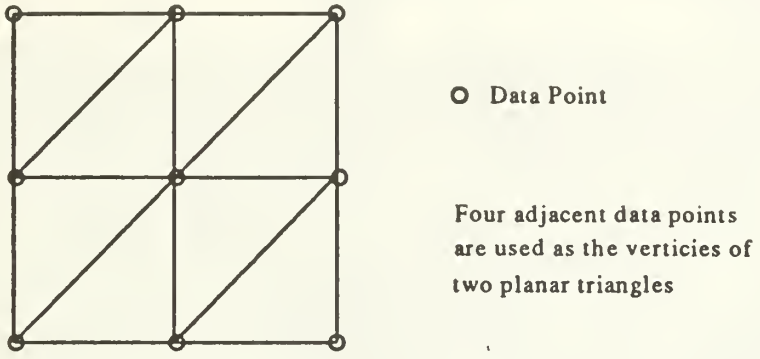
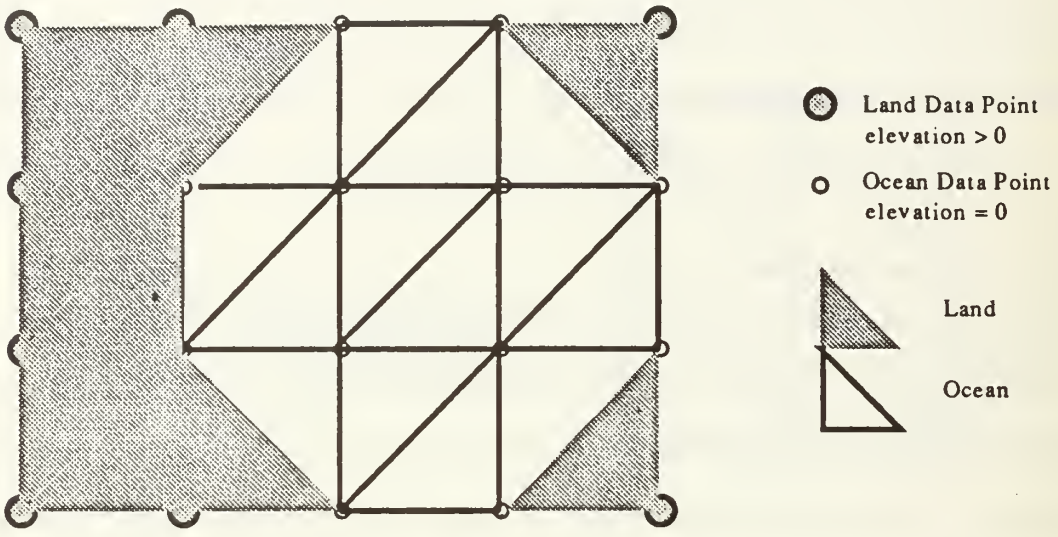


Figure 5.3 Area Between Data Points as Planar Triangles

land and one water (see Figure 5.4). Figure 5.5 shows a picture of a typical shoreline drawn with this method. Procedure *drawpoly.c* in Appendix D correctly draws the triangles for four adjacent points given as parameters.

C. THREE LAYERS OF RESOLUTION

Drawing terrain out to the horizon requires too many polygons when all the data points are used. The natural perspective of a three-dimensional scene causes objects in the distance to appear small and indistinct. A three dimensional display has this same property. A polygon drawn at a distance might map to only one pixel in screen coordinates. Because of this difference, objects in the background do not need to be drawn in the same resolution as those in the foreground. The terrain data is converted to three resolutions. The foreground is drawn in level three resolution, data points 100 yards apart. This is the same resolution as the original data. The second level of resolution is



All four points water both triangles are water
Three of the points are water one triangle land one water
Any other combination both triangles are land

Figure 5.4 Shoreline Polygons

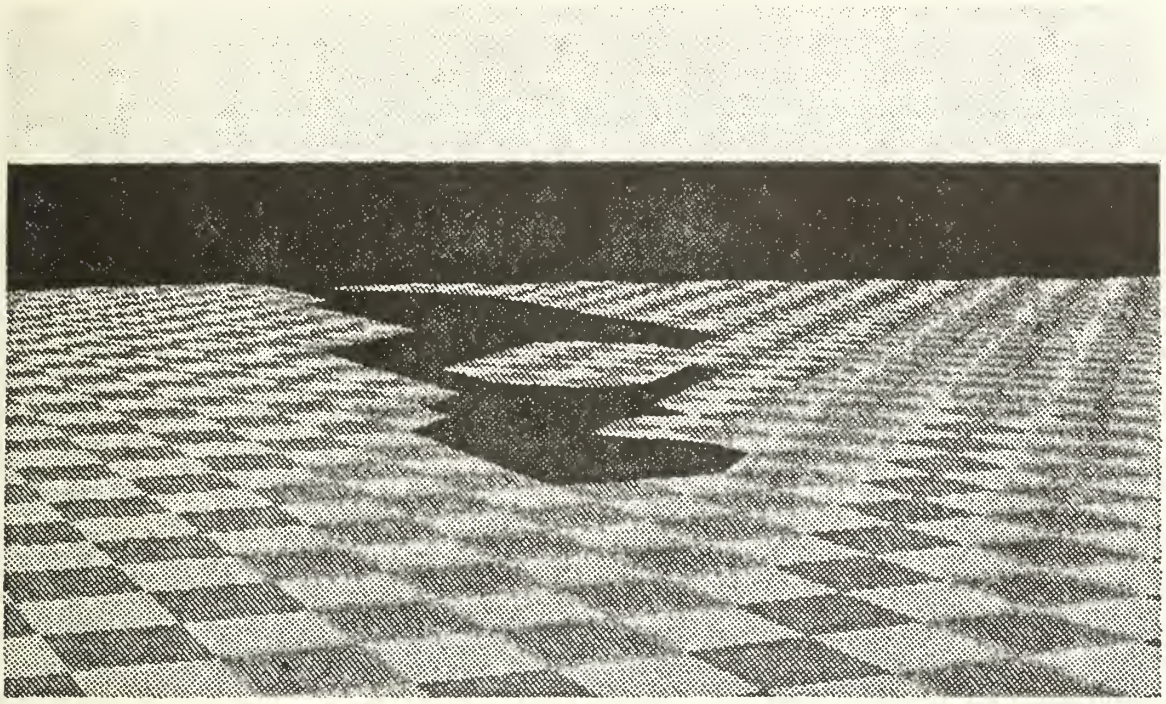


Figure 5.5 Sample Shoreline

for the middle-ground. Data points are 1200 yds apart. This resolution has one data point for every 144 level one points. Level one data is for drawing distant terrain. There are 100 level two points for one level one point and the data points are 12,000 yards, six NM, apart. This data can be preprocessed to fit one of two proposed data structures. By using different resolutions to draw out different distances, the number of polygons can be cut by an order of magnitude without degrading the display. Figure 5.6 shows two pictures, the first is drawn completely with high resolution, the second with a three-tiered structure.

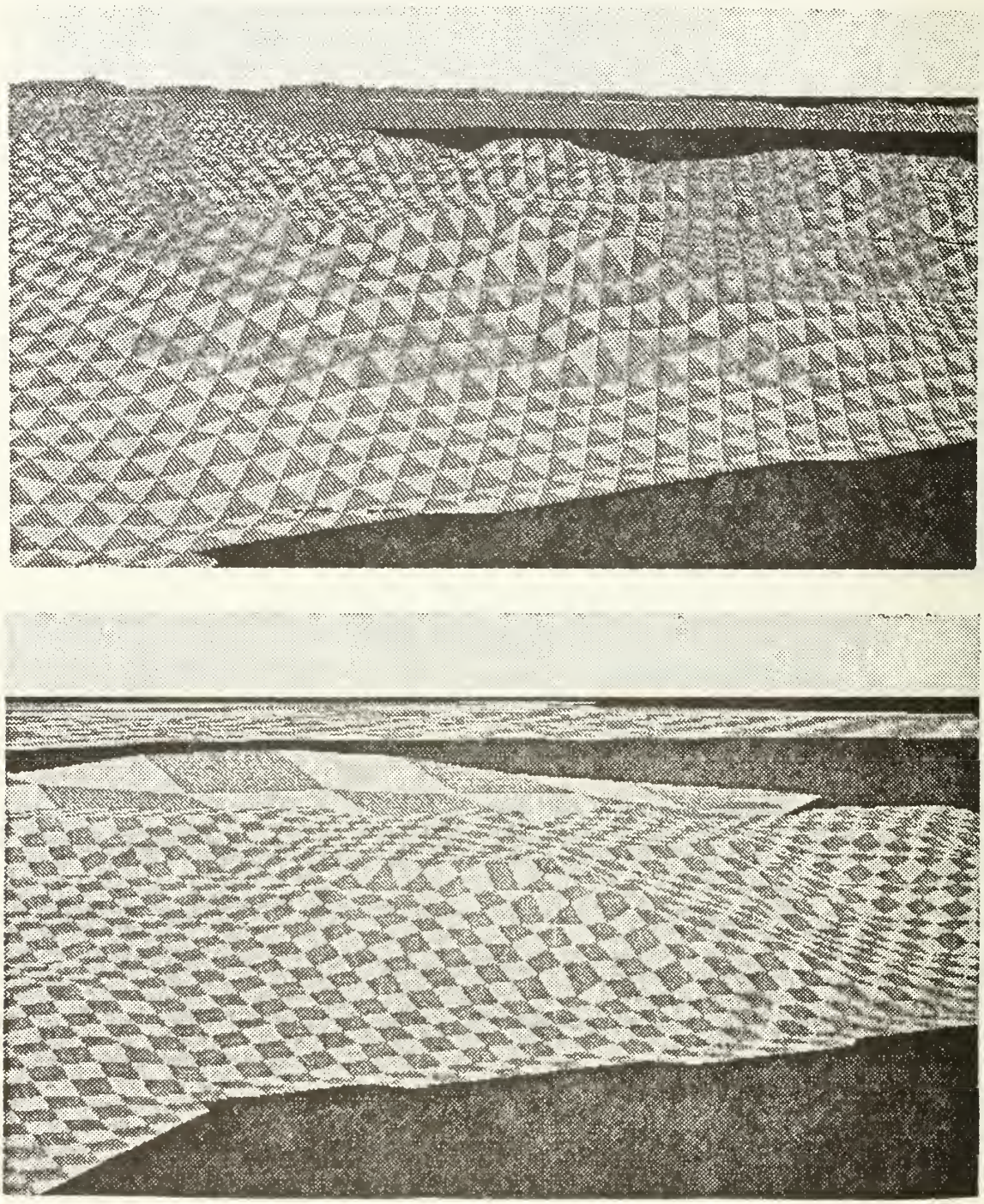


Figure 5.6 Contrast Single and Multi Resolution

1. Implementation

Large amounts of data are required to accurately display terrain in a three-dimensional view. Access to the data needs to be fast if near real-time animation is expected. The data structure that is used should be optimal in both space and time. Two possible data structures are examined to hold the terrain data required for three tiers of resolution. One uses a hierarchical structure with pointers. This structure also compresses the data. The other uses three two-dimensional arrays and simplifies the program to display the terrain at the cost of more storage space. There are problems with the transition between different resolutions and the use of z-buffering to display the terrain over the water base. Most of the problems have been solved but work is still needed to refine the techniques and speed up the algorithm. This section is written to explain the implementation to readers who might not be familiar with the "tricks" commonly used in the C language.

a. The World

The database used in this study is divided into cells. Each cell is an area one degree latitude by one degree longitude. More then one cell can be displayed at one time and as the view point moves the system must be able to change cells. The earth can be divided into octants. Each octant consists of 8100 cells in a 90 by 90 array indexed by latitude and longitude. This implementation uses one octant indexed from 0 to 90 degrees north latitude and 90 to 180 degrees east longitude. This matrix is sparsely filled with only 30 cells in the database. The array starts out as all null pointers. Pointers to individual cell data structures are added as required. The drawing routine checks the array for a needed cell. If the pointer is null, the routine *init_terrain*[7] is called with the

[7]Appendix F, Appendix G

latitude and longitude of the requested cell as parameters. This procedure checks to see if the appropriate file is in the database. If not, the cell is assumed to be all ocean with zero elevation at all data points.

b. Multiple Cells

The procedure *draw_terrain*[8] first calculates the area that can be viewed in the scene. Draw_terrain then sets the view bounds, maximum and minimum coordinates in the X and Z direction. The coordinates of the view bounds are relative to the lower left corner of the cell containing the view position. The cell containing the view point is always drawn. Any view bounds that fall outside the cell area indicate that another cell must be drawn. There are 12 possible cases that are checked with a maximum of four cells that are drawn for any scene. Figure 5.7 shows an example where four cells will be drawn. The procedure *draw_cell*[9] is called once for each cell. The position of that cell relative to the cell with the view position is passed as a parameter and is used to adjust the view bounds to the coordinates of the new cell. The procedure *adjust_bounds*[10] is called by *draw_cell* to do the conversion.

c. The Cell

All the actual data for all three resolutions is stored in cells. The data structure of the cell is the controlling factor that dictates the efficiency of the drawing algorithms.

---

[8]Appendix E

[9]Appendix F, Appendix G

[10]Appendix E

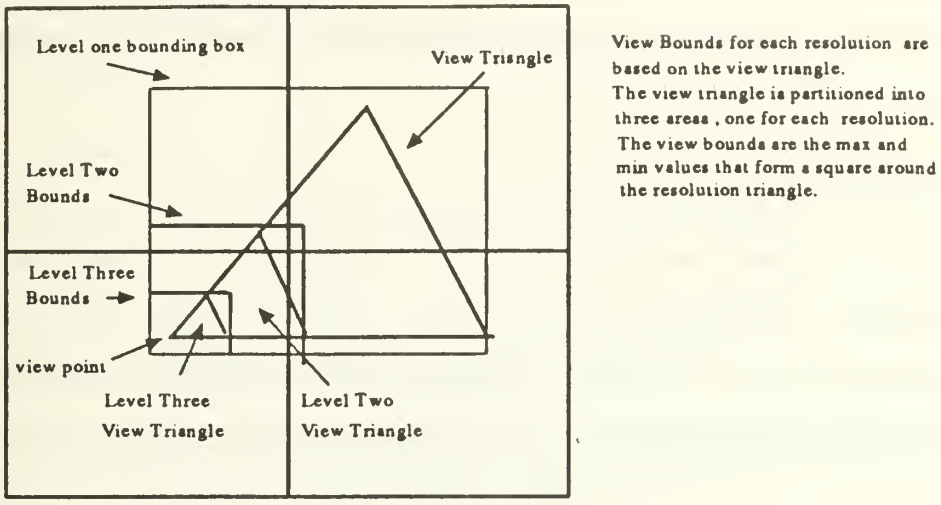


Figure 5.7 Multiple Cells Drawn With View Bounds

(1) Initialize the Structure. The data structure for cell data starts out empty. When data from a particular cell is first requested, the data must be read from the database and inserted into the appropriate structure. The display must pause and wait for the data to be entered. The first attempt to read just the raw data into an array took four minutes. This is clearly unacceptable. There are a couple of ways to speed-up this process. Block reads and self buffering improve the performance, cutting the time down to seven seconds.

In order for block reads to be effective, the database needs to be in the same form as the structure it is being read into. The initial data works very well when assumptions are made about the way the programming language stores the data. The initial data consists of sequential lists of data points. Each list has 1201 two byte integers listed sequentially according to ascending latitude. The lists are then stored in order according to longitude. The C language stores arrays in this same manner. The name of

an array is the address of the first element in the array. The elements in the array are stored sequentially in adjacent memory locations. The following line of code read 1201 two byte words from an input file and stores them sequentially starting at the address of Array_name.

```
nbyte = fread(Array_name,2,1201,fdi);
```

In this way a whole array can be filled with one read. This also works with two-dimensional arrays. Array[2], of a two dimensional array, is the address of the starting location of the third column in the array. Experiments were conducted to determine the most effective buffer size to read this particular data. A buffer size of 28672 bytes was chosen.

The data needs to be in a form that conforms to the data structure before it is read. Additional preprocessing can also aid the performance. For instance, the conversion of data from meters to yards can be accomplished at run time or calculated and stored, ready for the program to read. Polygon normals, used for lighting, are another example of information that can be preprocessed.

(2) Drawing the Terrain. Only the procedures that actually access the elevation data are dependent on the data structure. The procedure that controls the drawing of the scene is *draw_terrain()*[11] This procedure performs the structure independent tasks and calls the procedure *draw_cell()*[12] to access the data and cause the proper polygons to be drawn. *Draw_cell()* is structure dependent and is covered in the following sections on the specific data structures. The procedure *draw_terrain* :

---

[11] Appendix E

[12] Appendix F or Appendix G depending on the data structure

* Calculates the visibility
* Sets the drawing perspective
* Draws the sky
* Draws the ocean to the horizon
* Calculates the area to be drawn in each of the three resolutions
* Determines what cells to draw
* Calls the procedure draw_cell() for each of the cells

Most of the work done by this procedure is in calculating the view bounds that tell the procedure *draw_cell()* what part of the cells to display.

(3) Array Structure. Two-dimensional arrays have properties that make them an ideal structure for this application. Each data point can be indexed according to its world coordinate location and is accessed in constant time without traversing a complicated data structure. The following structure is used to store the data for each cell.

```
typedef struct{
        short level3[1201][1201];
        short level2[101][101];
        short level1[11][11];
        } cell;
```

Each resolution is stored in a matrix that can be accessed separately. When a cell not already in memory is requested for a scene, the procedure *init_terrain*[13] allocates memory and reads the data into the structure.

---

[13]See Appendix F

The procedure *3a.c*[14] preprocesses the raw terrain data and writes the processed data to a file that can be easily read into the array structure. The data values for the different resolutions are calculated in this procedure. Level three resolution is the initial data. It is stored as a 1201 by 1201 array. Figure 5.8 is the code that calculates the values for the levels one and two data. The level two data is an array 101 by 101 points. Each value is the average of the level three points around it. The level one data uses the previously calculated level two data to further process the data into the coarser resolution of an 11 by 11 matrix. Because the data points at the cell boundaries are shared between cells, the level one and two points at the cell boundary must have the same value as the corresponding point in the adjacent cell. This is achieved by using the original data value for points around the boundary instead of the averages. The data is written to a sequential file in order to make reading the data quick and easy.

Init_terrain reads in the new data file in the same order it was written. The level three data is read first, one column at a time followed by the level two and level one data. This structure displays one new frame every four to five seconds on the IRIS 4D/70G workstation. This time depends on the amount of ocean in the scene. More ocean means fewer polygons and less time.

(4) Pointer Structure. The cells that are of interest in a typical naval situation include large areas of ocean containing data points with zero elevation value. Polygons are not be drawn for this data when an underlying plane is drawn to represent the ocean. It would be nice to preserve the direct access of the array structure and cut down the space required to store the data structure by not saving all the zeros. The following data structure is used to accomplish this.

---

[14] See Appendix F

```
/* Level Two Data Calculation */
for (i=0;i<=100;i++)
      for (j=0;j<=100;j++)
      {
/* cell boundary points get the value of the underlying point */
      /* if cell boundary */
         if((i==0)||(j==0)||(i==100)||(j==100))
            {
            level2[i][j]=level3[i*12][j*12];
            }
         else
         /* the level three points within 12 data points of the level two
             point are averaged to find the value of the level 2 point */
            {
            sub=0;/* subtotal */
            for(s=((i-1)*12);s<((i+1)*12);s++)
                  for(t=((j-1)*12);t<((j+1)*12);t++)
                  {
                  sub = sub + level3[s][t];
                  }
            level2[i][j] = sub/576;/* final value avg of 576 points checked */
            }
      }


/* Level One Data Calculation */
for (i=0;i<=10;i++)
      for (j=0;j<=10;j++)
      {
      /* if cell boundary actual value of underlying point*/
         if((i==0)||(j==0)||(i==10)||(j==10))
               {
               level1[i][j]=level2[i*10][j*10];
               }
         else
         /* level 1 value is average of underlying level 2 values */
            {
            sub=0;/* subtotal */
            for(s=((i-1)*10);s<((i+1)*10);s++)
                  for(t=((j-1)*10);t<((j+1)*10);t++)
                  {
                  sub = sub + level2[s][t];
                  }
            level1[i][j] = sub/400;/* final value avg of 400 points */
            }
      }
```

Figure 5.8 Calculation of Level One and Level Two Data

```
typedef struct{
        short       data[13][13];
        }level3_rec;
typedef level3_rec *ptr3;


typedef struct{
        ptr3        level3ptr[11][11];
        short       level3val[11][11];
        short       all_zero;
        }level2_rec;
typedef level2_rec *ptr2;

typedef struct{
        ptr2        level2ptr[11][11];
        short       level2val[11][11];
        short       all_zero;
        }level1_rec;
typedef level1_rec *ptr1;
```

Each level of data has a separate structure. The octant that points to the individual cells holds pointers to only the level one structure. The level one and level two structures have two arrays. One contains the actual data values for the resolution while the other contains pointers to the structures for the next level. If an underlying structure is all zeros, the structure is not allocated and the pointer is null (see Figure 5.9).

Preprocessing data for this structure is more complex. The data is first read into a 1201 by 1201 array. This data is then used to fill in the data structure. The level one and two data value can be calculated as either the minimum or the average of the points below it in structure. As the data structure is filled, any subcell with all zero values is marked not to be saved. The program *3T.c* initially uses the raw data to create the data structure and then writes it to a file in an order that can be quickly read into the structure while the program is executing.

The level two and three data is basically converted from one large array to many smaller arrays. The cell boundaries share data points with adjacent cells. This duplicated data is needed in each cell to allow polygons to be drawn to the edge of the cell. The same is true of the sub-cells of level two and three data. The data points on

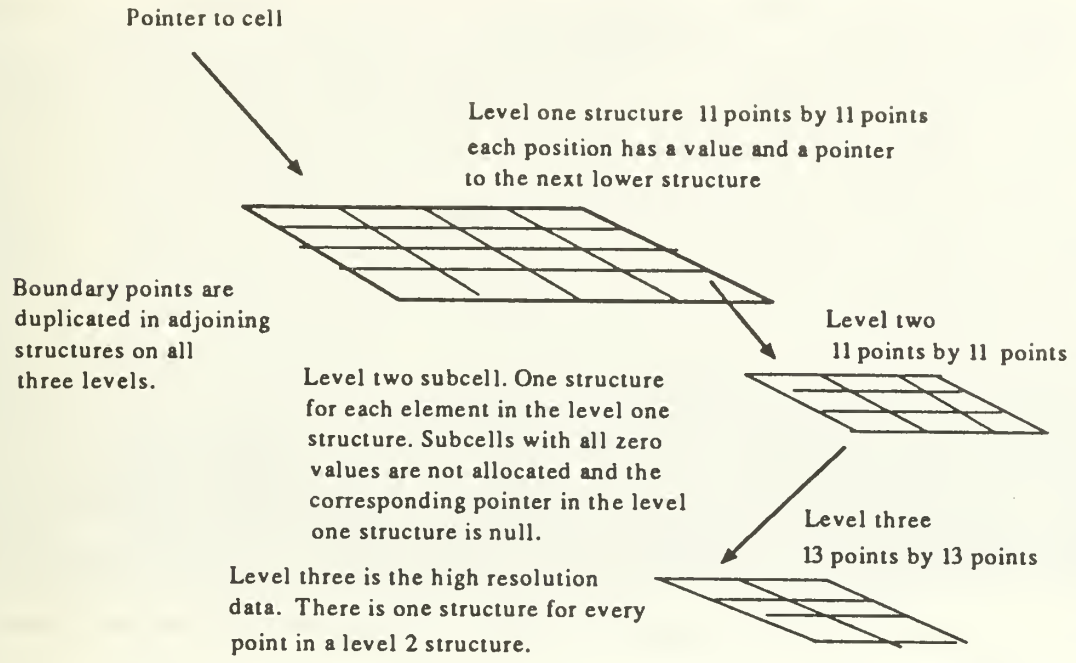


Figure 5.9 Pointer Data Structure

the edge of the sub-cells must be duplicated in the adjacent subcells and the values must be equal. This is done when the data is initially set up in the program *3T.c*.

The values for the level one and level two data were calculated and displayed as both the mean value and the low value. There was no discernible difference in the display. Since the level one and two polygons are drawn in the distance, a small difference in altitude is not significant.

d. Resolution Transitions

The transition between resolutions is not smooth. The level one transition to level two is in the distance and not noticeable but the line between level two and three

shows noticeable gaps where the level three data meets the level two polygon at its end points but not at the level three points in between.

This was solved by drawing a "skirt" around the level two boundary with level three. The procedure *draw_skirt*[15] draws a vertical green plane from the two points passed in as a parameter to zero elevation. This procedure is called from the procedure *draw_cell* when a level two polygon is drawn that boarders level three. (see Figure 5.10)

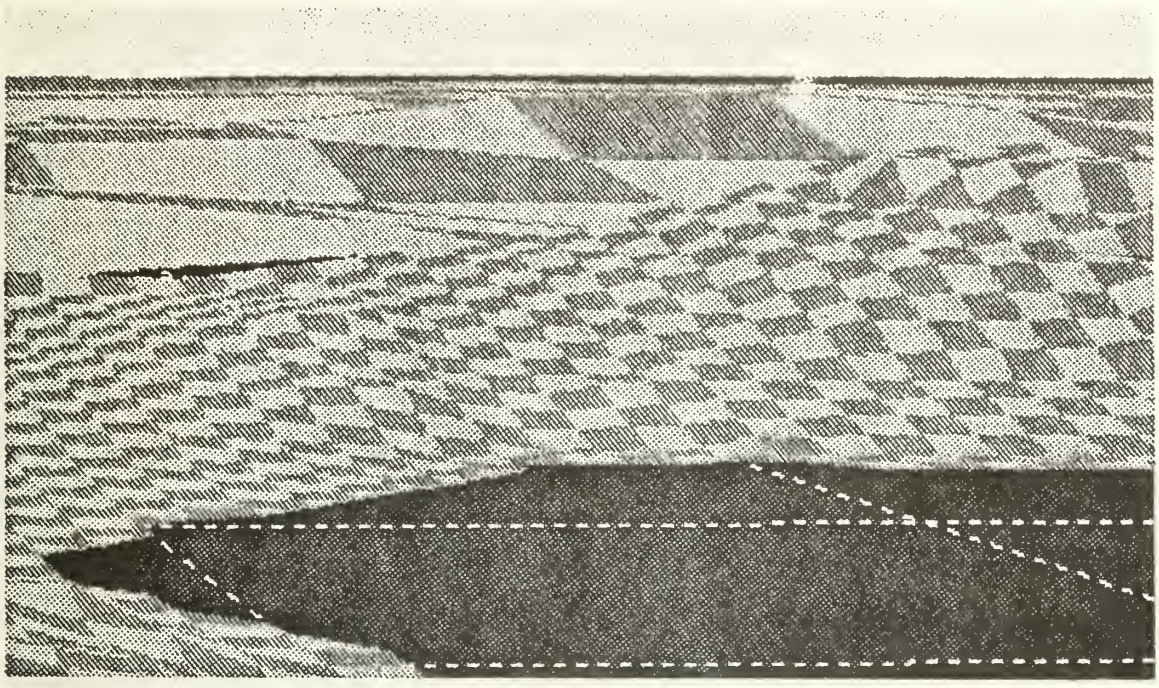
e. Z-buffering

The use of z-buffering as a hidden surface technique has caused several problems in this implementation. The first problem is speed. Each time a pixel is to be drawn to fill a polygon its z value must be compared with the pixel value already in the z-buffer. If the z value in the buffer is closer then the new point, the buffer stays the same and the new point is not drawn. In this way a pixel is only changed if the new pixel value is in front of the old one. This comparison each time a pixel is written is very expensive and time consuming but the algorithm is simplified because the order that the polygons are drawn is unimportant. New hardware is supposed to greatly increase the speed of z-buffering to a point where it is much faster then trying to keep track of the individual polygons. A quick port of the terrain display to the IRIS 4D/GT resulted in an order of magnitude improvement.

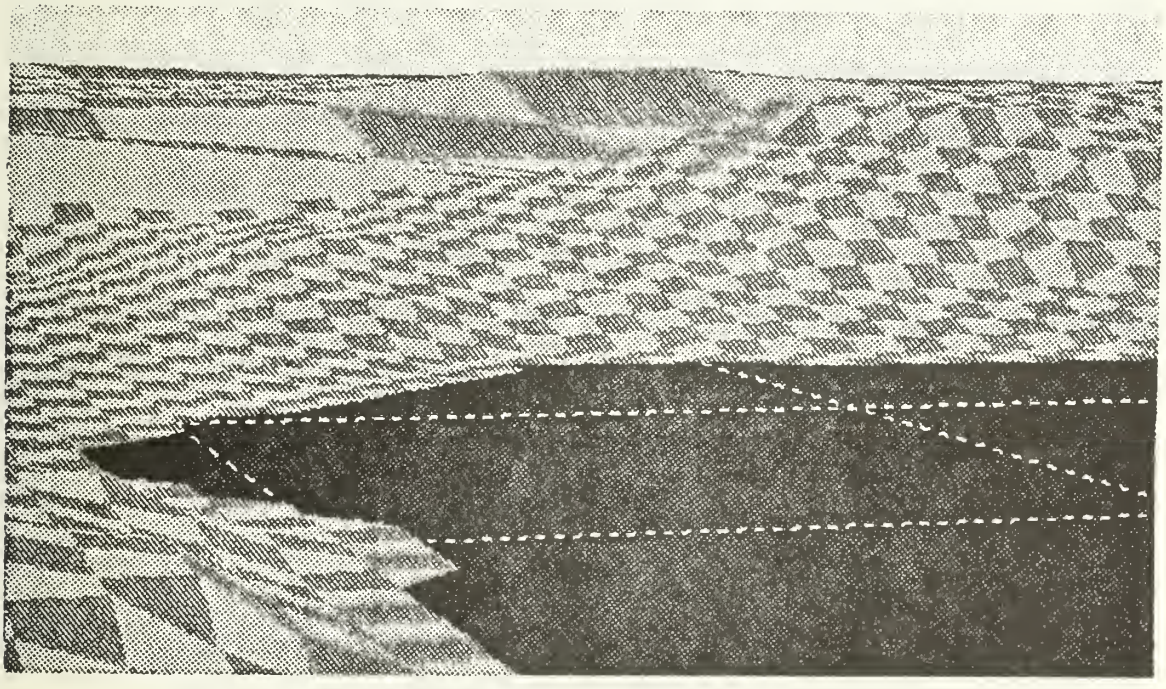
Z-buffering caused another problem that affected the resolution boundaries. You cannot draw one resolution over another resolution and depend on the high resolution polygons always being shown. The low resolution data will cover low spots or valleys in the high resolution terrain. Since the boundaries cannot overlap, the

---

[15]Appendix D



Resolution Boundary Gap



Skirt Drawn To Fill The Gap

Figure 5.10 Resolution Boundary

lower resolution must be drawn out to the boundary of the next higher resolution. The bottom line is that more polygons must be drawn, and the program must do more calculation to compute the boundaries.

Z-buffering does not always work correctly when two parallel planes are drawn close to each other. When the terrain is drawn over one blue polygon representing the ocean, the shoreline and the ocean are basically two parallel planes that often map to the same pixel and z-buffer location. The pixel value that is drawn is unreliable. The effect is popping blue splotches on the terrain as the view point changes. Figure 5.11 shows two pictures of the same area taken at different elevations. As you can see, the shape of the shoreline changes and is not regular.

There are two solutions to this problem that don't require that more polygons be drawn. Drawing the ocean before or after the terrain has no effect. The z-buffering seems to work better close to the near clipping plane. Moving the near clipping plane forward as far as possible helped some but did not solve the problem. Another help was changing the far clipping plane to be one and a half times the visibility. This in effect decreases the resolution in the z direction but the picture quality is improved slightly. It was suggested that at least 20 bitplanes are required for the z-buffer [Ref. 13]. The IRIS 4D/70G has 24 bitplanes but due to a software problem only 16 can be accessed. The new IRIS 4D/GT can now access all 24 bit-planes. Preliminary tests on the new machine indicate that the z-buffer resolution is not the cause of this particular problem. The final part of this solution was to lower the water a distance that is about one pixel below the land. This causes enough separation to insure that the land is drawn over the water. A distance of six yards worked well for the level two and three data. There is still some problems in the background but the overall picture is good (see Figure 5.12).

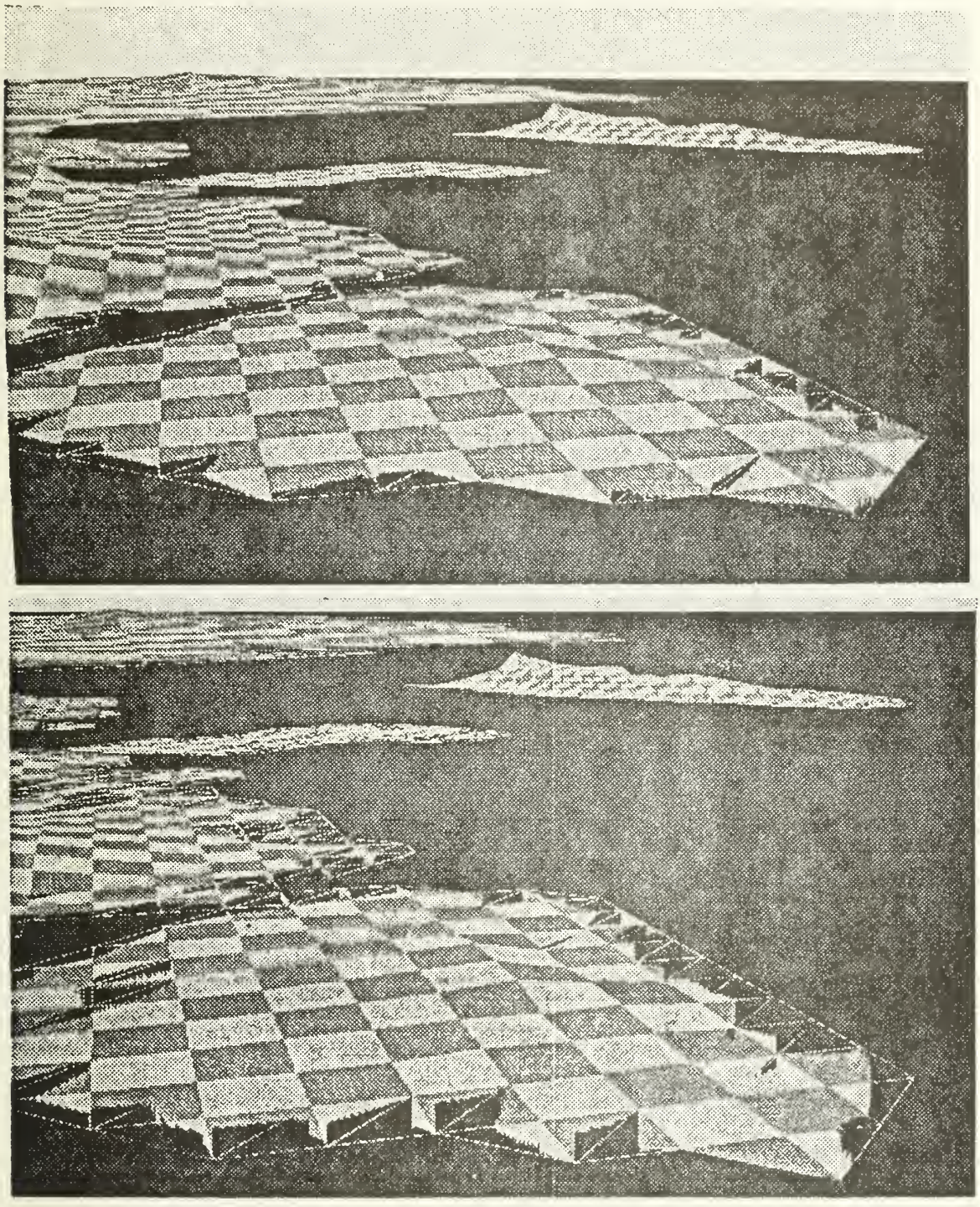


Figure 5.11 Z-buffer Changing Coastline

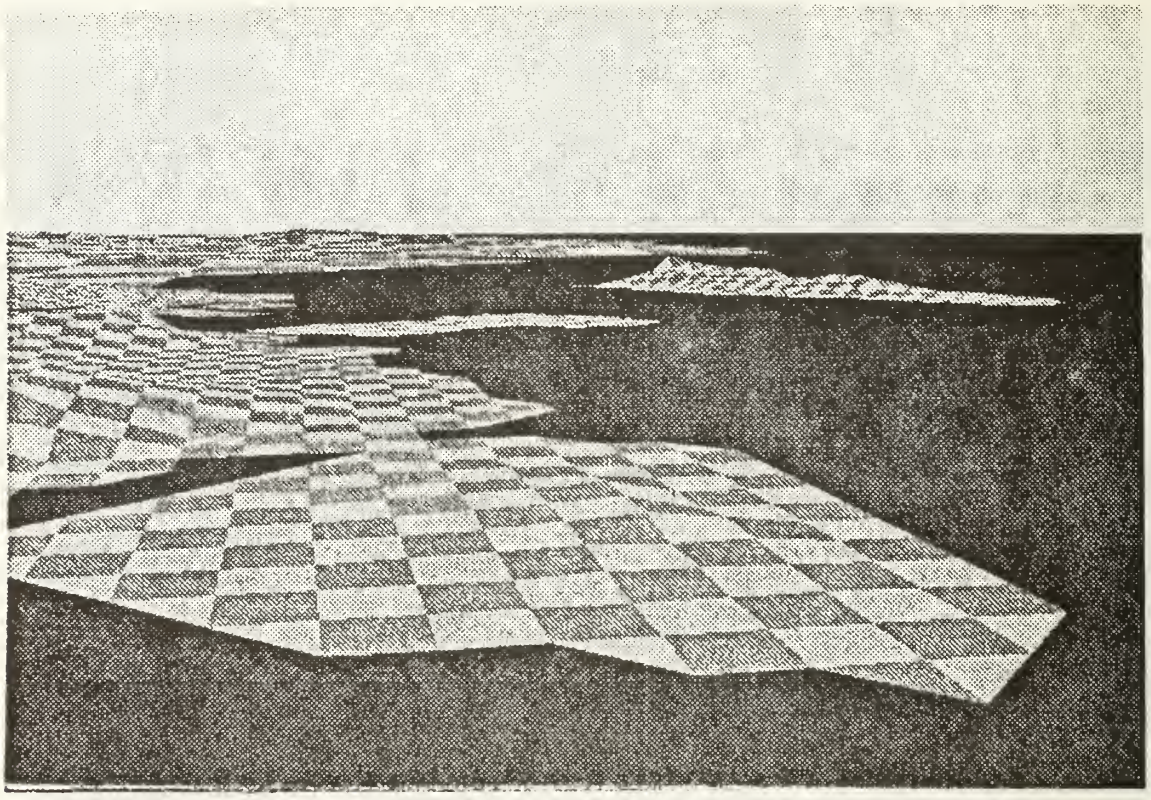


Figure 5.12 Picture With Corrections to Z-buffering

There is another way to solve this problem if you don't need hidden surface elimination for anything drawn under the water. Turn z-buffering off before drawing the ocean plane. Then turn z-buffering on, clear the z-buffer, and draw the terrain. This insures that everything will be drawn over the ocean plane. This won't cause a problem unless the intention is to draw additional objects that might lie partially under water, i.e. ships.

Another solution to this problem requires that polygons be drawn for the ocean. This separates the sea and land so they are not drawn as two parallel planes. This solution takes more time to draw the extra polygons. If the hierarchical structure is used, the complexity of the program increases because of the null pointers. The program must

determine what size the polygons are and where to draw them when a null pointer is encountered. This is still better then drawing all the water polygons for the standard resolution because only one polygon needs to be drawn for each null pointer.

D. CONCLUSIONS

1. Data Structures

The Array structure requires about 3 megabytes of data. The pointer requires about 0.8 megabytes for an average cell, about half water half land. The procedures for the initialization and drawing the structure are much more complex with the pointer system. The time required to draw one frame is three to five seconds for the array and one to five for the pointers. At worst case, when the scene is all land, performance is about equal. The pointer system excels when the viewpoint is over water and null pointers are encountered when drawing. The performance of the hierarchical pointer structure is at worst the same as the array. The improvement when drawing both land and water make the increased complexity of the hierarchical pointer version of the program worthwhile.

2. Z-buffering

Z-buffering is not always the best solution to the hidden surface problem. Many of the problems with displaying the terrain would be greatly reduced if the painter's algorithm was used instead. If the end use of this research was to just display terrain the painters algorithm would be acceptable. The difficulty comes when you want to add objects to the display, i.e. ships, tanks, buildings, etc. The painter's algorithm relys on the drawing order of the polygons. The polygons closest to the viewpoint are drawn last covering any objects farther away. Any objects that are drawn in addition to the terrain must have their polygons sorted with the terrain polygons to insure that all the

polygons that represent the final picture are drawn in order, far to near. See [Ref. 7] for a better discussion of this technique. If the problems with z-buffering can be overcome, it is well worth the effort for a three dimensional display.

Black and white images are used throughout this chapter to illustrate points about the display. Figures 5.13 and 5.14 are color photographs of typical views seen in the three-dimensional terrain display. They are added to help equate the black and white images to the views actually seen on the system's color display.

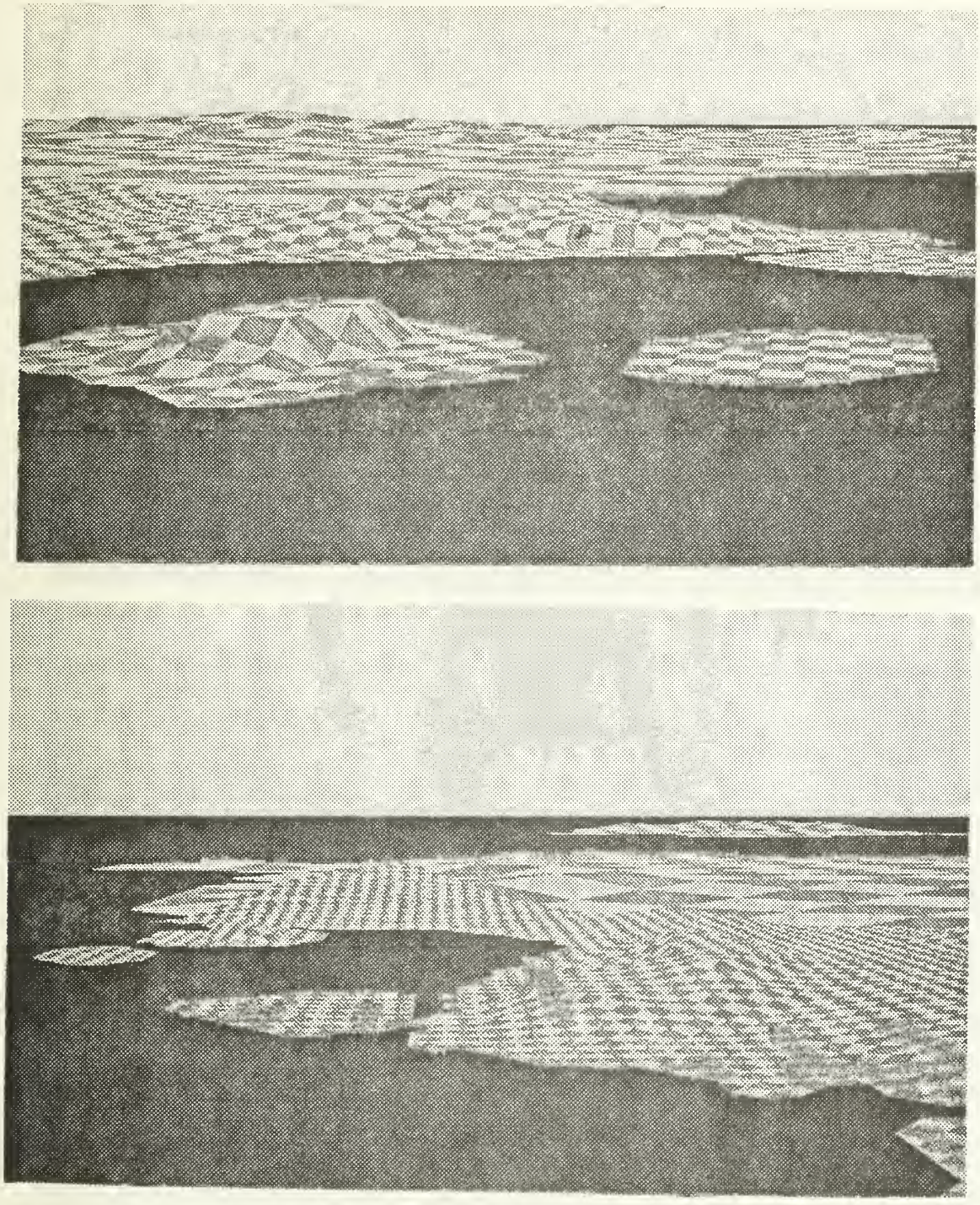


Figure 5.13 Color Photos of the Terrain Display

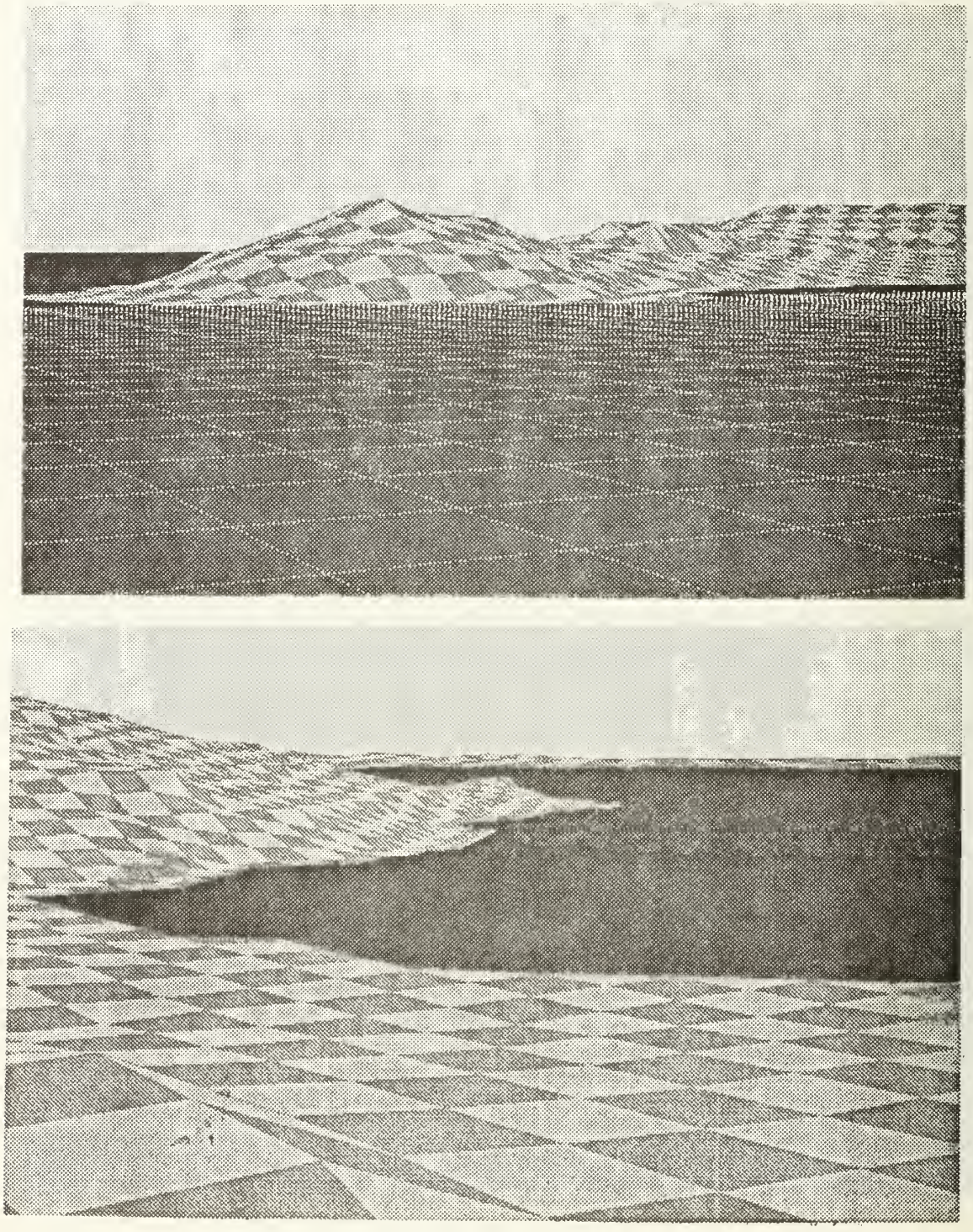


Figure 5.14 Color Photos of the Terrain Display

VI. CONCLUSIONS

A commander's workstation that provides useful information in a way that is easy to understand and control is not only feasible but essential for managing the vast amount of information a commander needs to make speedy, well informed decisions. This study focuses on two preliminary issues in the development of the command and control workstation of the future (CCWF), a user interface with multiple windows and a three-dimensional display of the sea/land environment.

A. THE USER INTERFACE

Multiple windows is a necessity. Separate, physical displays for each type of information is not practical. The use of multiple windows as virtual displays allows the user to quickly and easily arrange the screen to show the information that is required.

A mouse or track-ball controlled cursor is an easy way to control the entire workstation. The user can "point and click" to perform any operation without having to fumble with unfamiliar switches and dials.

Controls that are presented on the screen in pop-up menu form are easier to understand and execute. After the selections have been made, the menu disappears until it is needed again. The menus are changed according to the situation. Only the executable subset of actions is presented to the user. This gives the user fewer choices to sort through before making a selection. The selections are presented on the screen in simple easy to understand English. Operations that are infrequently executed are presented in a separate menu that is called from the main menu. This is an effective way

to change default system settings. Menu driven controls increase the effectiveness of the user interface with the CCWF.

B. THREE-DIMENSIONAL DISPLAY

A three-dimensional display is very complex. It requires a very powerful computer capable of drawing on the order of 100,000 polygons per second. This study focuses on the display of digital terrain elevation data in a three-dimensional display.

Data from the Defense Mapping Agency standard tapes is used. Appendix C shows the routines developed to extract files from the DMA standard tapes and also the routines developed to extract the needed information from the files.

The terrain is drawn by passing elevations for four adjacent points to a routine that draws two triangles representing the area between the four points. An acceptable approximation of the actual shoreline can be generated by assuming that any square with three of the points at zero elevation has one triangle land and one water. Any square with all zero elevations is all water and any other combination is all land.

Objects in the distance are seen in less resolution then objects that are close. A technique that models this natural fact is to use three resolutions of terrain data. A high resolution data with points close together, is used to draw terrain in the foreground. A medium resolution has points that are farther apart and a low resolution is used to draw things far away.

A hierarchical structure with pointers works very well to store three resolution terrain data. The data is stored in small structures starting with low resolution on top and working down three layers to the high resolution data. The value of each low resolution point is based on the values of all the higher resolution points below it in the structure. This technique also compresses the data. A command and control workstation for the

Navy is primarily concerned with the area of sea and land. Many of the data values for these cells are zero in elevation, indicating ocean. Any sub cell with all zero values is not stored. The structure that represents this area is not allocated and the pointer to this section is null. When a null pointer is encountered while drawing the terrain the system knows all values below this point are zero.

Z-buffering was used as the means of hidden surface elimination. This method makes drawing many polygons very easy but is very costly in time. It is hoped that future hardware improvements will overcome the problems and allow z-buffering to live up to its potential.

C. FUTURE WORK

The next step in the development of CCWF is to combine the multiple window interface with the three-dimensional display. New hardware, the IRIS 4D/GT, provides the computation and graphics power needed to provide multiple, separately controlled windows, showing two and three-dimensional views in near real-time. The two-dimensional version of the command and control workstation needs to be ported to the new hardware and changed to conform more closely with the native window manager. The three-dimensional terrain display should be included in the new system. Enhancements needed in the two-dimensional display include information on the closest point of approach, track history and other common NTDS functions. The ability for the user to enter information directly into the system should also be added. This can be used to enter contacts into the system or show special marks or boundary areas on the display.

A new display showing a map of the region needs to be developed. Terrain elevation data is available and cultural data can be obtained from DMA. NTDS symbology should be shown on the display indicating contacts.

All three of these displays need to be tied together, sharing the same information. The database of contact information needs to be expanded to include a three-dimensional graphics representation that can be displayed and selected in the three-dimensional display. The view point of the three-dimensional display should be set by selecting a contact or a point in the NTDS or chart display.

Work is needed in the three-dimensional display to take advantage of the additional power provided in the IRIS 4D/GT. A lighting model should be added to provide more realistic views with Gouraud shading. Contacts, buildings and special features should be represented by special three-dimensional icons that resemble the actual objects. More accurate drawing bounds can be computed, cutting the number of polygons that are drawn.

The user should be able to attach to a contact and see the view from the perspective of someone on a ship's bridge or in a plane's cockpit. The view-point should also have the ability to be free floating, where the user selects a position and then moves about at will. The user selects a contact in formation to attach to. While attached, the view shown on the three-dimensional display is based on the position, course and the height of eye of the contact. The user then detaches from the contact. He can assign a course and speed to the view point and move about, observing the formation from any angle.

Tactical information, such as weapons and sensor envelopes, can be added to the three-dimensional display. The user selects a contact and asks for the weapons systems to be displayed. A window opens with a list of weapons and sensors. The user then asks to

see the envelopes of all or some of the systems. The area that is effected on the three dimensional display is overdrawn with a transparent color showing the effective area.

The information available in Defense Mapping Agency cultural files can be added to both the two and three-dimensional displays. The chart display can show features commonly found on maps and charts, i.e. cities, landmarks, buildings, towers, navigational aids, etc. The three-dimensional view can use three-dimensional icons to represent these same features.

A system with all of the features listed above is a major effort. In fact we foresee that even greater graphics capabilities are needed to completely carry out our desires for the three-dimensional display. Fortunately, we see near future graphics workstations that will give us an order of magnitude increase in polygons per second in the next two years. Before that time we need to prototype and explore three-dimensional capabilities with the hope that we are able to provide a prototype of the command and control workstation of the future, as the new hardware arrives.

APPENDIX A - DEFENSE MAPING AGENCY TAPES

Tape 1	**Tape 2**
six cells	six cells
30N 129E	32N 128E
30N 130E	32N 129E
30N 131E	32N 130E
31N 129E	32N 131E
31N 130E	32N 132E
31N 131E	32N 133E

Tape 3	**Tape 4**
eight cells	nine cells
33N 126E	33N 125E
33N 127E	33N 126E
33N 128E	33N 127E
33N 129E	33N 128E
33N 130E	33N 129E
33N 131E	33N 130E
33N 132E	33N 131E
33N 133E	33N 132E
	33N 133E

APPENDIX B - DIGITAL TERRAIN ELEVATION DATA FORMAT

This is the file discription for the Defense Mapping Agency digital terrain elevation data used in this project. It was extracted from the Defence Mapping Agency publication "Product Specifications for Digital Landmass System (DLMS) Database", stock number SPEXDLMS2.

CHAPTER 4 -- DIGITAL FILE DESCRIPTIONS
SECTION 100 -- TERRAIN

101. General

The DMA Standard Terrain Format is DMA's standardized system of recording terrain elevation data on magnetic tape. The format is intended for the purposes of production, storage and exchange of terrain elevation data.

102. File Characteristics

A. Physical Characteristics of Magnetic Tape

1. Length: 2400 feet

2. Width: .5 inch

3. Nine track recording format

4. Odd parity

5. Density/recording method:

a. 1600 FPI/Phase encoded. This is preferred by DMA for storage and data exchange and will normally be expected from generators and provided to requestors.

b. 800 FPI/NRZI. This is not preferred by DMA but will be provided to or accepted from data requestors or generators unable to accept or generate 1600 FPI, phase encoded data.

c. 6250 FPI/GCR. Future preferred DMA data exchange format. Will not be used unless agreed to by sender and receiver.

6. Inter-Record gap: .6 inch (6250 FPI: .3 inch)

7. Physical end-of-tape markers at the beginning (beginning-of-tape marker) and end of the tape (end-of-tape marker).

B. Record Characteristics

1. Recorded Labels: American National Standard Magnetic Tape Labels for Information Interchange X3.27 - 1969. Recorded in ASCII code.

2. Data Records:

a. Record size: variable length, maximum 14414 frames, minimum 14 frames, modal (average) 2414 frames.

b. Blocking factor: 1:1 (block size = record size)

3. Record Sequence:

VOL 1 (Volume Header Label)
HDR 1 (File Header Label for file A)
UHL 1 (User Header Label for file A)

*

DSI (for file A)
ACC (for file A)
Data (for file A)

*

EOF 1 (End of File for file A)
UTL 1 (User Trailer Label for file A)

*

HDR 1 (File Header Label for file B)
UHL 1 (User Header for file B)

*

DSI (for file B)
ACC (for file B)
Data (for file B)

*

EOF 1 (End of File for file B)
UTL 1 (User Trailer Label for file B)

*
*

NOTE: In the above sequence, a Tape Mark (hardware end of file) is denoted by an "*".

4. Logical Characteristics: (Level 1)

a. Data File Structure: Arranged into 1 degree by 1 degree geographic areas. Each data file will contain data falling within a single one degree square. The reference origin for each data file will be the Southwest corner of the degree square. Multiple data files will be arranged primarily by ascending latitude bands (-90° South to +90° North), secondarily by ascending longitude (-180° West to +179° East).

b. File Extent: To provide overlap between adjacent data files, the degree square coverage in this standard includes the even degree values on all sides of the area. Each data record has one point of overlap with the square above and one with the square below (if the record extends to the degree square limits). Entire data records lying on integer degree longitude values will also exist in the adjacent degree square.

c. Terrain Elevation Intervals: The horizontal plane spacing of the elevation array will be in whole second intervals for intervals of 1 second and above and in 0.1 second intervals for intervals less than 1 second.

d. Data Value Sequence: The elevations within a data record have a constant longitude value. The first data value is the southernmost known elevation and the last is the northernmost. Unknown values internal to the record are indicated by the null state condition of all one-bits. No two data records will have the same longitude value.

e. Data Record Sequence: Within a data file, the records are arranged in order by ascending longitude.

f. Hash Control Total Information: The last four frames of each type data block contain a 32 bit value which is a checksum computed algebraically by summing all elevations and header words in that block, as 8-bit values using integer arithmetic. Each frame from tape is considered as an 8-bit value for checksum calculation.

5. Field Characteristics:

a. Numeric Value: All elevation values are signed magnitude binary integers, right justified, 16 bits. The sign is the high order position. Negative values are not complemented.

b. Permissible Elevation Value: $\pm$32767 meters

NOTE: This is the maximum allowable elevation value. However, this value will not exceed +9000 meters or -12,000 meters.

c. Null State Condition: Blank data will be all one bits.

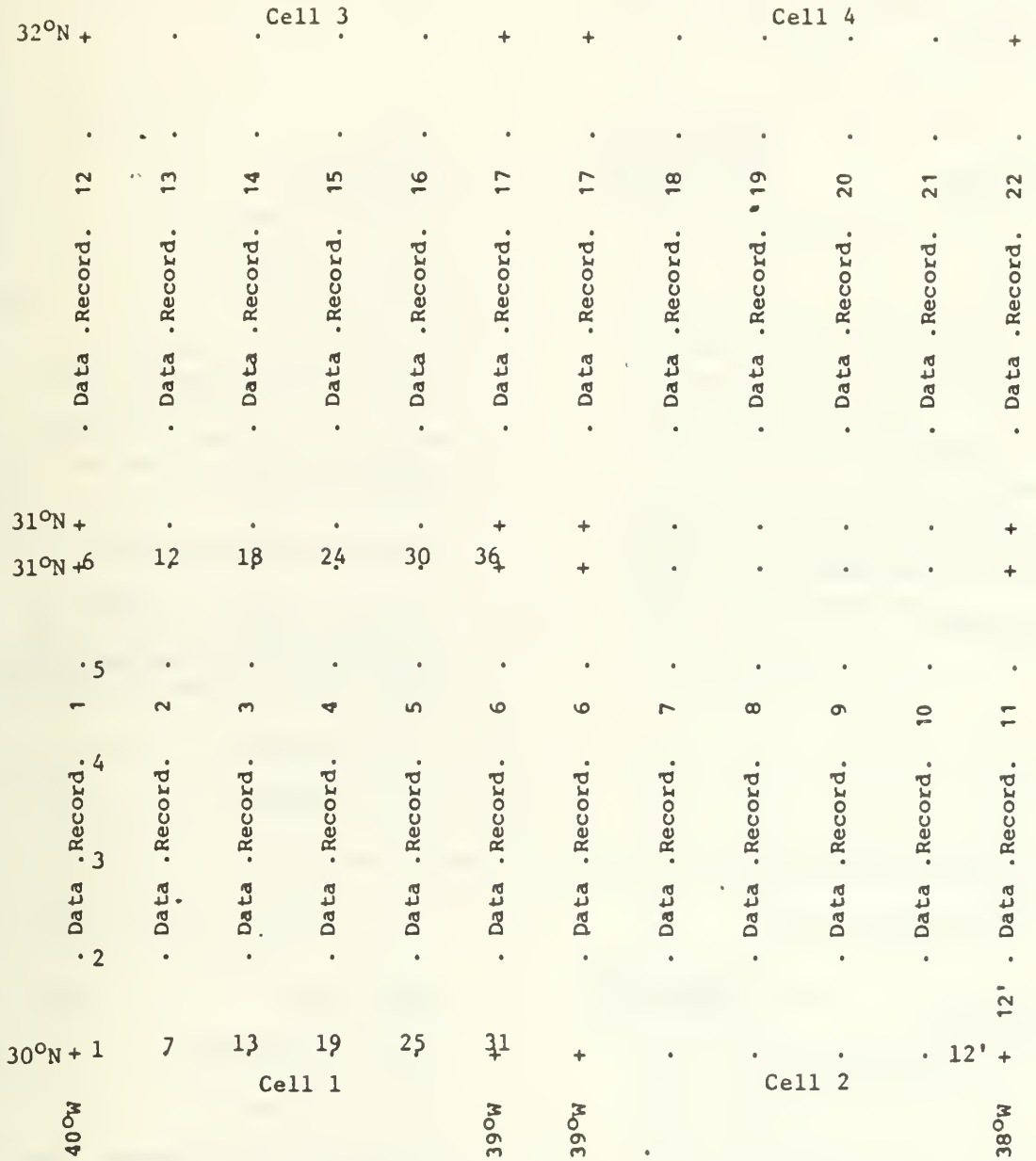
6. Explanatory Diagram: In order to more fully explain the file structure, figure 4-100-1 is included.

FIGURE 4-100-1
TERRAIN EXAMPLE
FOUR 1° CELLS
12' LONGITUDE SPACING
(NON STANDARD)

DATA STRUCTURE SEQUENCE	DATA STRUCTURE TYPE	DATA STRUCTURE SEQUENCE	DATA STRUCTURE TYPE
1	VOL 1	32	HDR 1
2	HDR 1	33	UHL 1 31°N, 40°W
3	UHL 1 30°N, 40°W	34	*
4	*	35	DSI
5	DSI	36	ACC
6	ACC	37	Data Record 12
7	Data Record 1	38	Data Record 13
8	Data Record 2	39	Data Record 14
9	Data Record 3	40	Data Record 15
10	Data Record 4	41	Data Record 16
11	Data Record 5	42	Data Record 17
12	Data Record 6	43	*
13	*	44	EOF 1
14	EOF 1	45	UTL 1
15	UTL 1	46	*
16	*	47	HDR 1
17	HDR 1	48	UHL 1 31°N, 39°W
18	UHL 1 30°N, 39°W	49	*
19	*	50	DSI
20	DSI	51	ACC
21	ACC	52	Data Record 17
22	Data Record 6	53	Data Record 18
23	Data Record 7	54	Data Record 19
24	Data Record 8	55	Data Record 20
25	Data Record 9	56	Data Record 21
26	Data Record 10	57	Data Record 22
27	Data Record 11	58	*
28	*	59	EOF 1
29	EOF 1	60	UTL 1
30	UTL 1	61	*
31	*	62	*

NOTE: * = Tape Mark

Figure 4-100-1
TERRAIN Example (Continued)



NOTES: (1) In above example (non-standard 12' Latitude/Longitude spacing), each Data Record contains 6 elevations. A 3" standard 1° square Data Record contains 1201 elevations.

(2) Elevations along 1° boundaries are repeated for each 1° square.

103. Record Formats

Digital Terrain Elevation Data. See Section 104 for further explanation of Records and Fields.

In the following record formats, a character requires one frame or 8 binary bits.

A. VOL Header Label

Field Contents	Field Length In Characters	Description
VOL	3	Recognition sentinel
1	1	Fixed by standard
	6	Reel Number Six alphanumeric characters identifying the physical reel
Blank or Nonblank	1	*Nonblank indicates restricted access, as the tape reel is privately owned
Blanks	26	Unrequired available space
Account Number	14	*Account number of owner of this tape reel (DMA uses a maximum of 12 characters left-justified, space filled)
Blanks	28	Fixed by standard
1	1	Fixed by standard

*These fields, to be defined by the producer, may be left blank.

B. HDR Header Label

Field Contents	Field Length In Characters	Description
HDR	3	Recognition sentinel
1	1	Fixed by standard
Filename	17	*Left-justified filename. The first 12 characters are referenced by the Executive System for comparison with the filename portion of the external filename.

Field Contents	Field Length In Characters	Description
UNIVAC	6	*Fixed as set identifier when referenced by system.
0001	4	*Reel sequence number within a file.
0001 - NNNN	4	*File sequence number within a reel.
0001 00	4 2	*Generation and version numbers which are fixed at 1 and 0.
bYYDDD	6	Creation date of tape. A blank followed by two characters for the year followed by three characters for the day (001 through 366) within the year. (date tape was written)
bYYDDD	6	*Expiration date of tape. Same format as creation date field. The date after which this tape reel may be considered as available for reallocation.
A space indicates unlimited access to this reel 15_8 - This reel is catalogued (on tape). 35_8 - This reel is catalogued with read key. 55_8 - This reel is catalogued with write key. 75_8 - This reel is catalogued with read and write key.	1	*Accessibility
Block Count	6	*Fixed at zeros.
Qualifier	13	*Used by the Executive Operating System (DMA uses a maximum of 12 characters left-justified space filled).
Blanks	7	Fixed by Standard.

*These fields to be defined by the producer may be left blank.

C. User Header Label

Field Contents	Field Length In Characters	Description
UHL	3	Recognition sentinel
1	1	Fixed by standard
DDDMMSSH	8	Longitude of origin (lower left corner of 1° Square-full degree value). H is the Hemisphere of the data.
DDDMMSSH	8	Latitude of origin (lower left corner of 1° Square-full degree value). H is the Hemisphere of the data.
SSSS	4	Longitude data interval in seconds (Decimal point is implied after third integer).
SSSS	4	Latitude data interval in seconds (Decimal point is implied after third integer).
* 0000-9999 or NA	4	Absolute Vertical Accuracy in meters. With 90% assurance that the linear errors will not exceed this value relative to mean sea level. (Right justified)
T - Top Secret S - Secret C - Confidential U - Unclassified R - Restricted	3	Security Code (Left justified)
Unique reference number	12	*Unique reference number (provide pointer to file containing detailed file description)
Number of longitude lines	4	Count of the number of longitude (profiles) lines

Field Contents	Field Length In Characters	Description
Number of latitude points	4	*Count of the number of latitude points per longitude line. Since the current implementation allows for variable record size, this field has very limited use.
Multiple accuracy	1	0 - Single 1 - Multiple
Reserved	24	Unused portion for future use

*These fields to be defined by the producer may be left blank.

D. Data Set Identification (DSI) Record
Fixed Length = 648 characters (Bytes)
Each Character = 1 Tape Frame = 8 Bits

Field Contents	Field Length In Characters	Description
DSI	3	Recognition Sentinel
T - Top Secret S - Secret C - Confidential U - Unclassified R - Restricted	1	Security Classification Code
	2	Security Control and Release Marking. For DoD use only (DIAM 65-19)
	27	Security Handling Description Other security description
	26	Reserved for future use
DTED1 or DTED2	5	DMA Series Designator for product type
	15	Unique reference number (For producing nations own use or zero filled)
	8	Reserved for future use
01-99	2	Data Edition Number (01-99)
A-Z	1	Match/Merge Version (A-Z)
YYMM	4	Maintenance Date (Zero filled until used.)

Field Contents	Field Length In Characters	Description
YYMM	4	Match/Merge Date
	4	Maintenance Description Code (All zero filled)
CCAAABBB (Country Agency Branch)	8	Producer Code (DIA Country Codes used for first 2 characters)
	16	Reserved for future use
	9	Product Specification Stock Number (SPEXDLMS2)
* 00 or 01-99	2	Product Specification Amendment and Change Number
YYMM	4	Product Specification Date (Currently 8304)
MSL	3	Vertical Datum (Mean Sea Level)
WGS72	5	Horizontal Datum Code (World Geodetic System 1972)
	10	Digitizing Collection System
YYMM	4	Compilation Date (Most descriptive month/year)
	22	Reserved for future use
DDMMSS.SH	9	Latitude of origin of Data. H is the hemisphere of data.
DDDMMSS.SH	10	Longitude of origin of Data. H is the hemisphere of data.
DDMMSSH	7	Latitude - SW corner of bounding rectangle. H is the hemisphere of the data.
DDDMMSSH	8	Longitude - SW corner of bounding rectangle. H is the hemisphere of the data.

Field Contents	Field Length In Characters	Description
DDMMSSH	7	Latitude - NW corner of data, bounding rectangle. H is the hemisphere of the data.
DDDMMSSH	8	Longitude - NW corner of data, bounding rectangle. H is the hemisphere of the data.
DDMMSSH	7	Latitude - NE corner of data, bounding rectangle. H is the hemisphere of the data.
DDDMMSSH	8	Longitude - NE corner of data, bounding rectangle. H is the hemisphere of the data.
DDMMSSH	7	Latitude - SE corner of data, bounding rectangle. H is the hemisphere of the data.
DDDMMSSH	8	Longitude - SE corner of data, bounding rectangle. H is the hemisphere of the data.
DDDMMSS.S	9	Clockwise orientation of data with respect to true North (will usually be all zeros for DTED).
SSSS	4	Latitude interval in tenths of seconds between rows of elevation values (Decimal point is implied after third integer).
SSSS	4	Longitude interval in tenths of seconds between columns of elevation values (Decimal point is implied after third integer).
0-9999	4	Number of Latitude lines Actual count - Number of latitude points. (rows that contain data)
0-9999	4	Number of Longitude lines Actual Count - Number of longitude points (columns that contain data)

Field Contents	Field Length In Characters	Description
* 00 or 01-99	2	Partial Cell Indicator 00 = Complete 1° square 01-99 = % of coverage completed.
	101	Reserved for DMA use only.
	100	Reserved for producing nation use only.
	156	Reserved for future use.

E. Accuracy Description (ACC) Record
Fixed Length = 2700 characters (Bytes)
Each Character = 1 Tape Frame = 8 Bits

Field Contents	Field Length In Characters	Description
ACC	3	Recognition Sentinel
* 0000-9999 or NA	4	*Absolute Horizontal Accuracy of Product in meters. NA if not specified.
* 0000-9999 or NA	4	*Absolute Vertical Accuracy of Product in meters. NA if not specified.
* 0000-9999 or NA	4	*Relative Horizontal Accuracy of Product in meters. NA if not specified.
* 0000-9999 or NA	4	*Relative Vertical Accuracy of Product in meters. NA if not specified.
*	4	Reserved for future use.
*	1	Reserved for DMA use only.
*	31	Reserved for future use.
* 00 or 02-09	2	Multiple Accuracy Outline Flag 00 - no outline provided 02-09 = number of accuracy subregions per 1° square (maximum 9)

If Product has subregional accuracies, the overall accuracy of the product will be the worst accuracy.

Field Contents	Field Length In Characters	Description
		Start of Accuracy Subregion Description. Repeat to maximum of nine times. Blank fill all unused accuracy subregions.
* 0000-9999 or NA	4	Absolute Horizontal Accuracy of subregion in meters. NA if not specified.
* 0000-9999 or NA	4	Absolute Vertical Accuracy of subregion in meters. NA if not specified.
* 0000-9999 or NA	4	Relative Horizontal Accuracy of subregion in meters. NA if not specified.
* 0000-9999 or NA	4	Relative Vertical Accuracy of subregion in meters. NA if not specified.
* 03-14	2	Number of coordinates in accuracy subregion outline. (Maximum of 14 coordinate pairs. Coordinates are input clockwise. Implied closing from last to first coordinate pairs.)
		Start of Coordinate Pair Description. Repeat to maximum of fourteen times to outline subregion. Blank fill all unused accuracy subregions.
DDMMSS.SH	9	Latitude. H is the hemisphere of the data.
DDDMMSS.SH	10	Longitude. H is the hemisphere of the data.
		End Coordinate Pair Description
		End Accuracy Subregion Description
*	18	Reserved for DMA use only.
*	69	Reserved for future use.

F. EOF Trailer Label

Field Contents	Field Length In Characters	Description
EOF	3	Recognition Sentinel
1	1	Fixed by standard

(See HDR header label for remainder of EOF fields.)

G. UTL Trailer Label

Field Contents	Field Length In Characters	Description
UTL	3	Recognition Sentinel
1	1	Fixed by standard

(See user header label for remainder of UTL fields.)

H. Data Record Description

Each element is a true elevation referenced to mean sea level datum recorded to the nearest meter. The horizontal position is referenced to specific longitude-latitude locations in terms of the World Geodetic System (WGS), determined on each file by reference to the origin at the Southwest corner. The elements are evenly spaced in latitude and longitude at the interval designated in the user header label in South to North profile sequence.

Field Contents	Field Length In Characters	Description
252_8	1	Recognition sentinel
Data block count	3	Sequential count of the block within the file, starting with zero for the first block (Fixed Binary)
Longitude count	2	Count of the meridian. True longitude = longitude count X data interval + Origin (S.W. corner) (Fixed Binary)

Field Contents	Field Length In Characters	Description
Latitude count	2	Count of the parallel. True latitude = latitude count X data interval + origin (S.W. corner) (Fixed Binary)
Elevation 1	2	True elevation value of point 1 of meridian in meters (Fixed Binary)
Elevation 2	2	True elevation value of point 2 of meridian in meters (Fixed Binary)
Elevation N	2	True elevation value of point N of meridian in meters (Fixed Binary)
Checksum	4	Algebraic addition of contents of block. Sum is computed as an integer summation of 8-bit values (Fixed Binary)

NOTE: Fixed Binary denotes signed magnitude, right-justified binary integers.

104. EXPLANATION OF RECORDS AND FIELDS (DTED). The following explanation of Records and Fields supplements, wnere necessary, the descriptions shown in Section 103.

A. VOL Header Label.

This record is required for labeled tapes in accordance with ANSI standard X3.27-1969 Magnetic Tape Labels for Information Interchange.

B. HDR Header Label.

This record is required for labeled tapes in accordance with ANSI standard X3.27-1969 Magnetic Tape Labels for Information Interchange.

C. UHL User Header Label.

1. ANSI standard allows an optional user header label in the first file of a labeled tape. Several computer manufacturers have implemented tape labeling in such a way that the user header label in the first file of the tape is inaccessible. This record is maintained for minimum impact to users not desiring to use the DSI record, but all information in it is in the DSI record as well.

2. Fields.

a. Longitude of Origin - Origin is always a full degree value even though the format allows values to be expressed to the second.

b. Latitude of Origin - Origin is always a full degree value even though the format allows values to be expressed to the second.

c. Seconds Longitude Interval - A square of DTED is North-South oriented with columns of elevation posts running from south to north. The longitude interval is the East-West distance between the columns expressed as tenths of seconds.

d. Seconds Latitude Interval - The spacing between the elevation posts within a column (i.e., the distance between the rows) is the latitude interval.

e. Accuracy - The accuracy of the product in meters.

D. Data Set Identification (DSI) Record (DTED).

1. This record provides all identification and security information related to the product except accuracy information. It duplicates information from the Header Record so that users may process the data using only the information in the DSI record if desired.

2. Fields.

a. Security Control and Release Markings - The two character codes are from DIAM 65-19.

b. DMA Series Designator - Five character code identifying the product in DMA Area Requirements and Product Status (ARAPS) file.

c. Unique Reference Number - to be determined.

d. Data Edition Number - The number assigned to the data indicating either original compilation (Edition 1) or subsequent replacements of the data (Editions 2, 3, etc.) to achieve accuracy requirements (recompilation) or currency/specification requirements (revision). The data edition number does not reflect the number of replacements made to the data to effect boundary matches.

e. Match/Merge Version - The number of times an edition of the data was changed to effect boundary continuity with adjacent data in the Cartographic Data Base (CDB).

f. Maintenance Date - The date existing data was either revised (updated) to meet the currency requirements (or to effect specification changes), or recompiled to meet accuracy requirements. When the existing data is only revised (horizontal position or vertical values are not significantly changed) the maintenance date will reflect the date of the revision, but the compilation date will not be changed -- it will continue to reflect the date of the original compilation. However, when the data is subjected to a major recompilation, the Compilation Date and the Maintenance Date will both be changed to reflect the date of the recompilation.

g. Match/Merge Date - The latest date the data was changed to effect continuity with adjacent data. This data corresponds to the Match/Merge Version Code.

h. Maintenance Description Code - to be determined.

i. Producer Code - The first two characters (left justified) indicate the producing nation and are from DIAM 65-18 - Geopolitical Elements and Related Files. The last six characters are to be used at the discretion of the producer. Blanks are acceptable.

Belgium BE	Netherlands NL
France FR	Norway NO
Germany, Federal Republic of GE	United Kingdom UK
Italy IT	United States US

j. Product Specification Stock Number - Identifies the product specification containing the compilation and accuracy requirements used to produce the data. Currently SPEXDLMS2.

k. Product Specification Amendment and Change Number - Indicates the highest numbered amendment and change used to produce the data (Amendment 0, change 1 -- 01; Amendment 2, change 2 -- 22; etc.).

l. Vertical Datum Code - Currently MSL.

m. Horizontal Datum Code - Currently WGS72.

n. Digitizing Collection System - Identifies the equipment used to collect the cartographic values from the source material used, i.e., AGDS, LIS, UNAMACE.

o. Compilation Date - The date the data was either originally compiled (Edition 1) or the date existing data was subjected to a major recompilation which involved significant changes to the horizontal positions and vertical values. (Edition 2, 3, 4, etc.)

* p. Latitude of Origin - Expressed in degrees, minutes, seconds and tenths of seconds with N or S to indicate hemisphere.

* q. Longitude of Origin - Expressed in degrees, minutes, seconds and tenths of seconds with E or W to indicate hemisphere.

E. Accuracy Description Record.

The accuracy record gives the accuracy of the product. The record allows space for the delineation of up to nine accuracy regions within the product should the accuracies of various portions of the product differ. Each outline may have up to fourteen coordinate pairs. Coordinates are input clockwise. The record is a fixed length record. Unused coordinate pairs are blank filled.

APPENDIX C - ROUTINES TO USE DMA DIGITAL TERRAIN DATA

The two programs in this appendix are for manipulating Defense Maping Agency digital terrain elevation data. The first, dted.c, reads one file at a time from the standard DMA magnetic tape. The name of the output file is passed as a command line argument. The program is repeatedly called with new names until the entire tape has been read.

The second program takes a data file extracted from the DMA tape and extracts the just the data from the file and writes it to a file whos name is the Latitude and longitude of the southwest corner of the cell.

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* FnName:      dted.c
* Author:      FRANK HARRIS
* Date:        jan 88
* Purpose:     to read dma dted tapes onto isiv1
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

#include <sys/file.h>
#define maxsize 3000
#define FALSE 1
#define TRUE  0
main(argc,argv)
int argc;
char *argv[];
{
      int fdi, fdo, nbyte;
      char buf[maxsize];
      short endfile ;

      /* open tape */
      /* tape not reset after each call */
      /* will open the tape where it was last stopped */
      if((fdi=open("/@isiv8/dev/snrmt0",O_RDONLY)) < 0)
      {
            printf("cannot open tape0);
            exit();
      }

      /* open output file, filename passed into program as argv param */
      if((fdo=creat(argv[1], 0666)) < 0)
      {
            printf("cannot open data file0);
            endfile = FALSE;
            exit();
      }
```

```
        /* read the next file file on the tape */
        while(!endfile)
        {
          nbyte=read(fdi,buf,maxsize);
          if(nbyte < 0)
              printf("read error0);
          if(nbyte == 0)
          {
              printf("end of file0);
              endfile = TRUE;
              exit();
          }
        nbyte = write(fdo,buf,nbyte);
        }
        close (fdi);
        close (fdo);
}
```

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* FnName:      extract.c
* Author:      FRANK HARRIS
* Date:        jan 88
* Purpose:  to extract terrain data from DMA file
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#define maxsize 3000
#define FALSE 0
#define TRUE  1
#define buf_size 28672
char *BUF,*BUFO;
int row,i,j;
char *malloc();
FILE      *fdi,*fdo;

main(argc,argv)
int argc;
char *argv[];
{
      int  j,i,n,nbyte;
      char name[9];
      char c[2];
      char buf[maxsize];
     /* open input file */
      if((fdi=fopen(argv[1],"r")) < 0)
      {
            printf("cannot open file0);
            exit();
      }

      /* set up input buffer */
      if ((BUF=malloc((buf_size )+1)) == NULL)
      {
       fprintf(stderr,"out of memory0);
       exit(1);
      }
      setvbuf(fdi,BUF,_IOFBF,buf_size );

      /* skip to lat long */
      fseek(fdi,185,0);
      /* extract lat long for filename */
```

```
/* use bytes 185,186,193,194,195,196,203 */
c[1]=' ';
nbyte=fread(c,1,1,fdi);
name[0]=c[0];name[1]=' ';
nbyte=fread(c,1,1,fdi);
strcat(name,c);
fseek(fdi,193,0);
nbyte=fread(c,1,1,fdi);
strcat(name,c);
nbyte=fread(c,1,1,fdi);
strcat(name,c);
nbyte=fread(c,1,1,fdi);
strcat(name,c);
nbyte=fread(c,1,1,fdi);
strcat(name,c);
fseek(fdi,203,0);
nbyte=fread(c,1,1,fdi);
strcat(name,c);

/* skip to first data record */
fseek(fdi,3348,0);

/* open output file */
if((fdo=fopen(name,"w")) < 0)
{
      printf("cannot open output file0);
      exit();
}
/*  allocate space for buffered I/O  */
if ((BUFO=malloc((buf_size )+1)) == NULL)
{
 fprintf(stderr,"out of memory0);
 exit(1);
}
setvbuf(fdo,BUFO,_IOFBF,buf_size );

for(i=1;i<=1201;i++)
{
   /* skip recog sym and record header */
   nbyte=fread(buf,1,8,fdi);
   /* read & write data */
   nbyte=fread(buf,1,2402,fdi);
   nbyte=fwrite(buf,1,2402,fdo);
   /* read checksum */
   nbyte=fread(buf,1,4,fdi);
```

```
    }
    fclose (fdi);
    fclose (fdo);
}
```

APPENDIX D - ROUTINES TO DRAW TERRAIN POLYGONS

These routines are used to draw the actual polygons. They are independent of the data structure used to store the terrain data.

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* FnName:      make_polly.c
* Author:      FRANK HARRIS
* Purpose:     to draw two triangles that are the correct
               orientation and color for the terrain display
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

make_polly(col,i,j,a,b,c,d,level_size)

short col;/* a flag to indicate what color to draw square */
int i,j;/* world coord location of lower left point "a" */
int a,b,c,d;/* elevation points */
int level_size;/* tells what size each side of the polygon is */
{
if(col==1)
  color(GREEN1);
else
  color(GREEN2);

if ((a==0)&&(b==0)&&(c==0)&&(d==0))
  {
  }
else if ((a!=0)&&(b==0)&&(c==0)&&(d==0))
  {
  pmvi(i,a,j);
  pdri(i,b,j-level_size);
  pdri(i+level_size,d,j);
  pclos();
  }
else if ((a==0)&&(b!=0)&&(c==0)&&(d==0))
  {
  pmvi(i,a,j);
  pdri(i,b,j-level_size);
  pdri(i+level_size,c,j-level_size);
  pclos();
  }
else if ((a==0)&&(b==0)&&(c!=0)&&(d==0))
  {
  pmvi(i,b,j-level_size);
```

```
  pdri(i+level_size,c,j-level_size);
  pdri(i+level_size,d,j);
  pclos();
  }
else if ((a==0)&&(b==0)&&(c==0)&&(d!=0))
  {
  pmvi(i,a,j);
  pdri(i+level_size,c,j-level_size);
  pdri(i+level_size,d,j);
  pclos();
  }
else
  {
  pmvi(i,a,j);
  pdri(i,b,j-level_size);
  pdri(i+level_size,c,j-level_size);
  pclos();
  pmvi(i,a,j);
  pdri(i+level_size,c,j-level_size);
  pdri(i+level_size,d,j);
  pclos();
  }
}
```

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* FnName:      draw_skirt.c
* Author:      FRANK HARRIS
* Purpose:     this routine is used to draw a vertical
               plane to be used as a gap filler between
               resolutions
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

draw_skirt(x1,z1,y1,x2,z2,y2,cell_size)
int x1,z1,y1,x2,z2,y2,cell_size;
{
if((y1!=0)||(y2!=0))
  {
  color(GREEN1);
  pmvi(x1*cell_size,y1,z1*cell_size);
  pdri(x2*cell_size,y2,z2*cell_size);
  pdri(x2*cell_size,0,z2*cell_size);
  pdri(x1*cell_size,0,z1*cell_size);
  pclos();
  }
}
```

APPENDIX E - ROUTINES TO DRAW THE TERRAIN

These routines calculate the structure independent information required to draw the terrain. The main procedure in this section is *draw_terrain()*. This proceudure controls the drawing process and calls the structure dependent code to actually draw the terrain. The second procedure *adjust_bounds()* is called by the structure dependent code to adjust the input parameters to match the cell coordinates for the particular cell being drawn.

```
#include "terrain.h"

octant myworld;
int   min4(),max4(),min3(),max3();

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* FnName:      draw_terrain.c
* Author:      FRANK HARRIS
* Date:        feb 88
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

/* 3d terrain display */


draw_terrain(view)
viewpoint view;
{
int i,j;

/* boundray points for three resolutions*/
/* farthest points of view triangle for each resolution
   the third vertex is the view position */
float level1x1,level1z1,level1x2,level1z2;
float level2x1,level2z1,level2x2,level2z2;
float level3x1,level3z1,level3x2,level3z2;
float ang1,ang2,vis;
float level1,ldir,lookx,looky,lookz;
float near,far;
/* bounding boxes for 3 levels  */
int max3x,max3z,min3x,min3z;
int max2x,max2z,min2x,min2z;
int max1x,max1z,min1x,min1z;
int gridx,gridz;

/* for timer routines */
long ttime,cpu,elapse;
struct tms ctime;
```

```
/* number of yards to the horizon */
vis = (VIS_COIF*sqrt(view.posy));
level1 = vis/SIN_FOV;
if (vis > view.max_vis)
   vis = view.max_vis;
/*************************************************************/
/* compute visability triangle */
/* triangle has verticies of current position and the line following
  the course out a distance of the computed visibility based on
  the height of eye and the area FOV degrees on either side of
  that line. */
ang1 = view.lookdir - FOV;
if (ang1 < 0.0)
  ang1 = ang1 + 360.0;
ang2 = view.lookdir + FOV;
if (ang2 > 360.0)
  ang2 = ang2 - 360.0;

/* compute points to draw to for all three levels */
level1x1 = view.posx + (level1 * sin(DtoR*ang1));
level1z1 = (-view.posz + (level1 * cos(DtoR*ang1)));
level1x2 = view.posx + (level1 * sin(DtoR*ang2));
level1z2 = (-view.posz + (level1 * cos(DtoR*ang2)));

level2x1 = view.posx + (LEVEL2 * sin(DtoR*ang1));
level2z1 = (-view.posz + (LEVEL2 * cos(DtoR*ang1)));
level2x2 = view.posx + (LEVEL2 * sin(DtoR*ang2));
level2z2 = (-view.posz + (LEVEL2 * cos(DtoR*ang2)));

level3x1 = view.posx + (LEVEL3 * sin(DtoR*ang1));
level3z1 = (-view.posz + (LEVEL3 * cos(DtoR*ang1)));
level3x2 = view.posx + (LEVEL3 * sin(DtoR*ang2));
level3z2 = (-view.posz + (LEVEL3 * cos(DtoR*ang2)));

/* find max and min values for bounding box on all three areas */
max1x = (max3(level1x1,level1x2,view.posx) / CELL_SIZE1);
max1z = (max3(level1z1,level1z2,-view.posz)/ CELL_SIZE1);
min1x = (min3(level1x1,level1x2,view.posx) / CELL_SIZE1);
min1z = (min3(level1z1,level1z2,-view.posz)/ CELL_SIZE1);

max2x = (max3(level2x1,level2x2,view.posx) / CELL_SIZE2);
max2z = (max3(level2z1,level2z2,-view.posz)/ CELL_SIZE2);
min2x = (min3(level2x1,level2x2,view.posx) / CELL_SIZE2);
min2z = (min3(level2z1,level2z2,-view.posz)/ CELL_SIZE2);
```

```
max3x = (max3(level3x1,level3x2,view.posx) / CELL_SIZE3);
max3z = (max3(level3z1,level3z2,-view.posz)/ CELL_SIZE3);
min3x = (min3(level3x1,level3x2,view.posx) / CELL_SIZE3);
min3z = (min3(level3z1,level3z2,-view.posz)/ CELL_SIZE3);

/*************************************************************/
pushmatrix();
/* compute vierwpoint for perspective */
looky = view.posy - 100.0*(tan(DtoR*view.lookang));
lookx = view.posx + 100.0*(sin(DtoR*view.lookdir));
lookz = view.posz - 100.0*(cos(DtoR*view.lookdir));

far  = vis * 1.5;
near = far/1000.0;

perspective(FOVY,ASPECT,near,far);
lookat(view.posx,view.posy,view.posz,lookx,looky,lookz,0);

color(SKYBLUE);
clear();

zbuffer(TRUE);
zclear();

/* start the timer */
set_timer(&ttime,&ctime);

/*************************************************************/
/* draw sea to horizon */
/* sea drawn 6 yards below horizon to solve z-buffer problem */
color(BLUE2);
pmv(view.posx,-6.0,view.posz);
pdr(level1x1,-6.0,-level1z1);
pdr(level1x2,-6.0,-level1z2);
pclos();
/*************************************************************/
/* draw grid lines */
if(view.gridlines == TRUE)
{
color(WHITE);
setlinestyle(SOLID);
linewidth(1);
if(view.posy > 300)
{
gridx = view.posx/CELL_SIZE1;
```

```
gridz = view.posz/CELL_SIZE1;
for (i=0;i<5;i++)
  {
  movei((gridx+i)*CELL_SIZE1,-2,(gridz+5)*CELL_SIZE1);
  drawi((gridx+i)*CELL_SIZE1,-2,(gridz-5)*CELL_SIZE1);
  movei((gridx-i)*CELL_SIZE1,-2,(gridz+5)*CELL_SIZE1);
  drawi((gridx-i)*CELL_SIZE1,-2,(gridz-5)*CELL_SIZE1);
  }
for (i=0;i<5;i++)
  {
  movei((gridx+5)*CELL_SIZE1,-2,(gridz+i)*CELL_SIZE1);
  drawi((gridx-5)*CELL_SIZE1,-2,(gridz+i)*CELL_SIZE1);
  movei((gridx+5)*CELL_SIZE1,-2,(gridz-i)*CELL_SIZE1);
  drawi((gridx-5)*CELL_SIZE1,-2,(gridz-i)*CELL_SIZE1);
  }
}
if(view.posy > 150)
{
gridx = view.posx/CELL_SIZE2;
gridz = view.posz/CELL_SIZE2;
setlinestyle(DASHED);
for (i=0;i<15;i++)
  {
  movei((gridx+i)*CELL_SIZE2,-2,(gridz+15)*CELL_SIZE2);
  drawi((gridx+i)*CELL_SIZE2,-2,(gridz-15)*CELL_SIZE2);
  movei((gridx-i)*CELL_SIZE2,-2,(gridz+15)*CELL_SIZE2);
  drawi((gridx-i)*CELL_SIZE2,-2,(gridz-15)*CELL_SIZE2);
  }
for (i=0;i<15;i++)
  {
  movei((gridx+15)*CELL_SIZE2,-2,(gridz+i)*CELL_SIZE2);
  drawi((gridx-15)*CELL_SIZE2,-2,(gridz+i)*CELL_SIZE2);
  movei((gridx+15)*CELL_SIZE2,-2,(gridz-i)*CELL_SIZE2);
  drawi((gridx-15)*CELL_SIZE2,-2,(gridz-i)*CELL_SIZE2);
  }
}

if(view.posy < 200)
{
gridx = view.posx/CELL_SIZE3;
gridz = view.posz/CELL_SIZE3;
setlinestyle(DOTTED);
for (i=0;i<30;i++)
  {
  movei((gridx+i)*CELL_SIZE3,-2,(gridz+30)*CELL_SIZE3);
```

```
  drawi((gridx+i)*CELL_SIZE3,-2,(gridz-30)*CELL_SIZE3);
  movei((gridx-i)*CELL_SIZE3,-2,(gridz+30)*CELL_SIZE3);
  drawi((gridx-i)*CELL_SIZE3,-2,(gridz-30)*CELL_SIZE3);
  }
for (i=0;i<30;i++)
  {
  movei((gridx+30)*CELL_SIZE3,-2,(gridz+i)*CELL_SIZE3);
  drawi((gridx-30)*CELL_SIZE3,-2,(gridz+i)*CELL_SIZE3);
  movei((gridx+30)*CELL_SIZE3,-2,(gridz-i)*CELL_SIZE3);
  drawi((gridx-30)*CELL_SIZE3,-2,(gridz-i)*CELL_SIZE3);
  }
}
}
/*************************************************************/
/* draw any terrain */
/* always draw the cell your in */
draw_cell(CELL,view.lat,view.longg,max1x,min1x,max1z,min1z,
                    max2x,min2x,max2z,min2z,
                    max3x,min3x,max3z,min3z);

if ((max1x > 10)&&(min1x < 0)&&(max1z >10))
  {
  draw_cell(ABOVE,view.lat+1,view.longg,max1x,min1x,max1z,min1z,
                    max2x,min2x,max2z,min2z,
                    max3x,min3x,max3z,min3z);
  draw_cell(RTUP,view.lat+1,view.longg+1,max1x,min1x,max1z,min1z,
                    max2x,min2x,max2z,min2z,
                    max3x,min3x,max3z,min3z);
  draw_cell(LTUP,view.lat+1,view.longg-1,max1x,min1x,max1z,min1z,
                    max2x,min2x,max2z,min2z,
                    max3x,min3x,max3z,min3z);
  }
else if ((max1x > 10)&&(min1x < 0)&&(min1z <0))
  {
  draw_cell(BELOW,view.lat-1,view.longg,max1x,min1x,max1z,min1z,
                    max2x,min2x,max2z,min2z,
                    max3x,min3x,max3z,min3z);
  draw_cell(RTDN,view.lat-1,view.longg+1,max1x,min1x,max1z,min1z,
                    max2x,min2x,max2z,min2z,
                    max3x,min3x,max3z,min3z);
  draw_cell(LTDN,view.lat-1,view.longg-1,max1x,min1x,max1z,min1z,
                    max2x,min2x,max2z,min2z,
                    max3x,min3x,max3z,min3z);
  }
else if ((max1z > 10)&&(min1z < 0)&&(max1x >10))
```

```
  {
  draw_cell(RIGHT,view.lat,view.longg+1,max1x,min1x,max1z,min1z,
                    max2x,min2x,max2z,min2z,
                    max3x,min3x,max3z,min3z);
  draw_cell(RTDN,view.lat-1,view.longg+1,max1x,min1x,max1z,min1z,
                    max2x,min2x,max2z,min2z,
                    max3x,min3x,max3z,min3z);
  draw_cell(RTUP,view.lat+1,view.longg+1,max1x,min1x,max1z,min1z,
                    max2x,min2x,max2z,min2z,
                    max3x,min3x,max3z,min3z);
  }
else if ((max1z > 10)&&(min1z < 0)&&(min1x <0))
  {
  draw_cell(LEFT,view.lat,view.longg-1,max1x,min1x,max1z,min1z,
                    max2x,min2x,max2z,min2z,
                    max3x,min3x,max3z,min3z);
  draw_cell(LTUP,view.lat+1,view.longg-1,max1x,min1x,max1z,min1z,
                    max2x,min2x,max2z,min2z,
                    max3x,min3x,max3z,min3z);
  draw_cell(LTDN,view.lat-1,view.longg-1,max1x,min1x,max1z,min1z,
                    max2x,min2x,max2z,min2z,
                    max3x,min3x,max3z,min3z);
  }
else if ((max1x > 10)&&(maxlz > 10))
  {
  draw_cell(ABOVE,view.lat+1,view.longg,max1x,min1x,max1z,min1z,
                    max2x,min2x,max2z,min2z,
                    max3x,min3x,max3z,min3z);
  draw_cell(RTUP,view.lat+1,view.longg+1,max1x,min1x,max1z,min1z,
                    max2x,min2x,max2z,min2z,
                    max3x,min3x,max3z,min3z);
  draw_cell(RIGHT,view.lat,view.longg+1,max1x,min1x,max1z,min1z,
                    max2x,min2x,max2z,min2z,
                    max3x,min3x,max3z,min3z);

  }
else if ((max1x > 10)&&(min1z < 0))
  {
  draw_cell(BELOW,view.lat-1,view.longg,max1x,min1x,max1z,min1z,
                    max2x,min2x,max2z,min2z,
                    max3x,min3x,max3z,min3z);
  draw_cell(RTDN,view.lat-1,view.longg+1,max1x,min1x,max1z,min1z,
                    max2x,min2x,max2z,min2z,
                    max3x,min3x,max3z,min3z);
  draw_cell(RIGHT,view.lat,view.longg+1,max1x,min1x,max1z,min1z,
```

```
                    max2x,min2x,max2z,min2z,
                    max3x,min3x,max3z,min3z);

  }
else if ((min1x < 0)&&(min1z < 0))
  {
  draw_cell(BELOW,view.lat-1,view.longg,max1x,min1x,max1z,min1z,
                    max2x,min2x,max2z,min2z,
                    max3x,min3x,max3z,min3z);
  draw_cell(LTDN,view.lat-1,view.longg-1,max1x,min1x,max1z,min1z,
                    max2x,min2x,max2z,min2z,
                    max3x,min3x,max3z,min3z);
  draw_cell(LEFT,view.lat,view.longg-1,max1x,min1x,max1z,min1z,
                    max2x,min2x,max2z,min2z,
                    max3x,min3x,max3z,min3z);

  }
else if ((min1x < 0)&&(max1z >10))
  {
  draw_cell(ABOVE,view.lat+1,view.longg,max1x,min1x,max1z,min1z,
                    max2x,min2x,max2z,min2z,
                    max3x,min3x,max3z,min3z);
  draw_cell(LTUP,view.lat+1,view.longg-1,max1x,min1x,max1z,min1z,
                    max2x,min2x,max2z,min2z,
                    max3x,min3x,max3z,min3z);
  draw_cell(LEFT,view.lat,view.longg-1,max1x,min1x,max1z,min1z,
                    max2x,min2x,max2z,min2z,
                    max3x,min3x,max3z,min3z);

  }
else if ((max1x > 10))
  {
  draw_cell(RIGHT,view.lat,view.longg+1,max1x,min1x,max1z,min1z,
                    max2x,min2x,max2z,min2z,
                    max3x,min3x,max3z,min3z);

  }
else if ((min1x < 0))
  {
  draw_cell(LEFT,view.lat,view.longg-1,max1x,min1x,max1z,min1z,
                    max2x,min2x,max2z,min2z,
                    max3x,min3x,max3z,min3z);

  }
else if ((max1z > 10))
```

```
  {
  draw_cell(ABOVE,view.lat+1,view.longg,max1x,min1x,max1z,min1z,
                  max2x,min2x,max2z,min2z,
                  max3x,min3x,max3z,min3z);

  }
else if ((min1z < 0))
  {
  draw_cell(BELOW,view.lat-1,view.longg,max1x,min1x,max1z,min1z,
                  max2x,min2x,max2z,min2z,
                  max3x,min3x,max3z,min3z);

  }



/*************************************************************/
zbuffer(FALSE);

popmatrix();

elapse = read_timer(&ttime,&ctime,&cpu);
printf(" time: elapse %d cpu %d",elapse,cpu);
}
```

```
/*************************************************************/
adjust_bounds(size,cell,maxx,minx,maxz,minz)
int size,cell,*maxx,*maxz,*minx,*minz;
{
if(*maxx>size)
 if ((cell==RIGHT)||(cell==RTUP)||(cell==RTDN))
   {
   *maxx = *maxx - size;
   *minx = 0;
   }
 else
   {
   *maxx = size;
   }
else
 if ((cell==RIGHT)||(cell==RTUP)||(cell==RTDN))
   {
   *maxx = 0;
   *minx = 0;
   }

if(*maxz>size)
 if ((cell==ABOVE)||(cell==RTUP)||(cell==LTUP))
   {
   *maxz = *maxz - size;
   *minz = 0;
   }
 else
   {
   *maxz = size;
   }
else
 if ((cell==ABOVE)||(cell==RTUP)||(cell==LTUP))
   {
   *maxz = 0;
   *minz = 0;
   }

if(*minx<0)
 if ((cell==LEFT)||(cell==LTUP)||(cell==LTDN))
   {
   *minx = *minx + size;
   *maxx = size;
   }
 else
```

```
    {
    *minx = 0;
    }
 else
  if ((cell==LEFT)||(cell==LTUP)||(cell==LTDN))
    {
    *maxx = 0;
    *minx = 0;
    }

 if(*minz<0)
  if ((cell==BELOW)||(cell==RTDN)||(cell==LTDN))
    {
    *minz = *minz + size;
    *maxz = size;
    }
  else
    {
    *minz = 0;
    }
 else
  if ((cell==BELOW)||(cell==RTDN)||(cell==LTDN))
    {
    *maxz = 0;
    *minz = 0;
    }
 }
```

APPENDIX F - MULTIPLE ARRAY DATA STRUCTURE

These are the structure dependent routines used with the multiple array data structure discussed in Chapter five. The procedure *3a.c* is used to preprocess the DMA digital terrain elevation data and place the processed data in a file format that is easily read by the application program. The terrain data is initially read into the application program's structure by the procedure *get_terrain()*. The final structure dependent procedure is *draw_cell()*. It is used to actually access the data structure and draw the visible terrain.

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* FnName:      3a.c
* Author:      FRANK HARRIS
* Date:        mar 88
* Purpose:     make 3 tier data structure file
               for display terrain.
               uses structure of three seperate
               arrays.
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#define maxelev 10000/* in yards higher the everest */
#define FALSE 0
#define TRUE  1
#define buf_size 28672
char *BUF,*BUFO;
char *malloc();
short  level3[1201][1201];
short  level2[101][101];
short  level1[11][11];

main(argc,argv)
int argc;
char *argv[];
{
     int  row,j,i,s,t,n,sub,nbyte;
     char *name;
     char c[2];
     short *P,lat,log;
     FILE      *fdi,*fdo;
    /* open input file */
     if((fdi=fopen(argv[1],"r")) < 0)
     {
```

```
            printf("cannot open file0);
            exit();
        }

/* set up input buffer */
        if ((BUF=malloc((buf_size )+1)) == NULL)
        {
         fprintf(stderr,"out of memory0);
         exit(1);
        }
        setvbuf(fdi,BUF,_IOFBF,buf_size );

/* read 1 colume of data at a time and put in an array
   the first index is the colume number "longitude"
   the second is for the row or latitude
   index [0][0] is the lower left corner of the cell*/
        for (i=0;i<=1200;i++)
        {
         P = &level3[i][0];
         nbyte = fread(P,2,1201,fdi);
        }

/* close input file no longer needed */
free(BUF);
fclose(fdi);

/* make output file name */
        name = argv[1];
        strcat(name,".3a");

        /* open output file */
        if((fdo=fopen(name,"w")) < 0)
        {
            printf("cannot open output file0);
            exit();
        }

/* setup output buffer */
        if ((BUFO=malloc((buf_size )+1)) == NULL)
        {
         fprintf(stderr," out of memory");
         exit(1);
        }
        setvbuf(fdo,BUFO,_IOFBF,buf_size );
```

```
/* convert lat long in file name to short int values */
      if (name_conv(&lat,&log,argv[1]))
      {
       printf(" filename not propper LAT/LONG");
       exit(1);
      }


/* start the file with lat long of lower left corner */
      nbyte=fwrite(&lat,2,1,fdo);
      nbyte=fwrite(&log,2,1,fdo);

for (i=0;i<=100;i++)
 for (j=0;j<=100;j++)
 {
/* cell boundary */
 if((i==0)||(j==0)||(i==100)||(j==100))
  level2[i][j]=level3[i*12][j*12];
 else
  {
  sub=0;
  for(s=((i-1)*12);s<((i+1)*12);s++)
   for(t=((j-1)*12);t<((j+1)*12);t++)
   {
   sub = sub + level3[s][t];
   }
  level2[i][j] = sub/576;
  }
 }

for (i=0;i<=10;i++)
 for (j=0;j<=10;j++)
 {
/* cell boundary */
 if((i==0)||(j==0)||(i==10)||(j==10))
  level1[i][j]=level2[i*10][j*10];
 else
  {
  sub=0;
  for(s=((i-1)*10);s<((i+1)*10);s++)
   for(t=((j-1)*10);t<((j+1)*10);t++)
   {
   sub = sub + level2[s][t];
   }
  level1[i][j] = sub/400;
```

```
    }
  }

/* write out the arrays */
 for (i=0;i<=1200;i++)
 {
  P = &level3[i][0];
  nbyte = fwrite(P,2,1201,fdo);
 }
 for (i=0;i<=100;i++)
 {
  P = &level2[i][0];
  nbyte = fwrite(P,2,101,fdo);
 }
 for (i=0;i<=10;i++)
 {
  P = &level1[i][0];
  nbyte = fwrite(P,2,11,fdo);
 }
/* clean up */
      fclose (fdo);
      free(BUFO);
}

/*********************************************************/
int name_conv(lat,log,name)
short *lat,*log;
char  *name;
{
short temp;
/* assume all lats and longs are north and east respectively */

temp=(short)name[0]-48;
if ((temp<0)||(temp>9))
   return(-1);
*lat = temp * 10;
temp=(short)name[1]-48;
if ((temp<0)||(temp>9))
   return(-1);
*lat = *lat + temp;

temp=(short)name[3]-48;
if ((temp<0)||(temp>9))
   return(-1);
*log = temp * 100;
```

```
temp=(short)name[4]-48;
if ((temp<0)||(temp>9))
   return(-1);
*log = ((temp*10) + *log);
temp=(short)name[5]-48;
if ((temp<0)||(temp>9))
   return(-1);
*log = (temp + *log);
return(0);
}
```

```
#include <string.h>
#include <sys/param.h>
#include <sys/types.h>
#include <sys/times.h>
#include "gl.h"
#include "device.h"
#include "constants.h"
#include "typedef.h"
#include "stdio.h"


#define maxelev 10000/* in yards higher the everest */
#define buf_size 28672/* for input buffer */

char *malloc();
char *BUF;

octant myworld;
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* FnName:      get_terrain.c
* Author:      FRANK HARRIS
* Date:        feb 88
* Amended:
* Purpose:
* Params:
* Returns:
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

get_terrain(lat,longg)
short  lat,longg;
{
int  row,j,i,n,nbyte;
char name[10],filename[100];
short buf[512];
FILE      *fdi;
cellptr ptr;
cell   P;
short *ptr1,*ptr2,*ptr3;

name_conv(lat,longg,name);
strcpy(filename,"/usr/work/fharris/3t_work/");
strcat(filename,name);
strcat(filename,".3a");

/* open input file */
```

```
if((fdi=fopen(filename,"r")) < 0)
{
 fprintf(stderr,"cannot open file0);
}
else
{
 /* set up input buffer */
 if ((BUF=malloc((buf_size )+1)) == NULL)
 {
  fprintf(stderr,"out of memory0);
 }
 else
 {
 setvbuf(fdi,BUF,_IOFBF,buf_size );


 /* get lat long first 2 items in file */
 fread(buf,2,2,fdi);

 /* make structure put pointer in world structure */
 myworld[lat][longg-90] = (cellptr)malloc(sizeof(cell));

 /* read in the arrays */
  ptr= myworld[lat][longg-90];
  for (i=0;i<=1200;i++)
  {
   ptr3 = &(ptr->level3[i][0]);
   nbyte = fread(ptr3,2,1201,fdi);
  }
  for (i=0;i<=100;i++)
  {
   ptr2 = &(ptr->level2[i][0]);
   nbyte = fread(ptr2,2,101,fdi);
  }
  for (i=0;i<=10;i++)
  {
   ptr1 = &(ptr->level1[i][0]);
   nbyte = fread(ptr1,2,11,fdi);
  }

 /* clean up */
 fclose (fdi);
 free(BUF);
 }
}
```

```
}

/*********************************************************/
/* pass in lat/long will return filename */

name_conv(lat,longg,name)
short lat,longg;
char  name[10];
{
/* assume all lats and longs are north and east respectively */
char    c[2];

c[1]= '  ';
name[1]='  ';
name[0] = (char)((lat/10)+48);
c[0] = (char)((lat%10)+48);
strcat(name,c);
c[0] = 'N';
strcat(name,c);
c[0] = (char)((longg/100)+48);
strcat(name,c);
longg=longg%100;
c[0] = (char)((longg/10)+48);
strcat(name,c);
c[0] = (char)((longg%10)+48);
strcat(name,c);
c[0] = 'E';
strcat(name,c);
}
```

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* FnName:      Draw_cell
* Author:      FRANK HARRIS
* Purpose:     Draws on e cell of terrain based on the
               input parameters.s version uses the
               multiple array data structure
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

draw_cell(cel,lat,longg,max1x,min1x,max1z,min1z,
                  max2x,min2x,max2z,min2z,
                  max3x,min3x,max3z,min3z)
short cel,lat,longg;
int   max1x,min1x,max1z,min1z;
int   max2x,min2x,max2z,min2z;
int   max3x,min3x,max3z,min3z;
{
int maxsub2x,maxsub2z,minsub2x,minsub2z;/* overlap bounds */
int maxsub3x,maxsub3z,minsub3x,minsub3z;
int stx1,stz1,spx1,spz1;/* start stop points for dif resolutions */
int stx2,stz2,spx2,spz2;
int stx3,stz3,spx3,spz3;
int i,j,s,t,u,v;
cellptr ptr;

/* get cell record */
if ((ptr = myworld[lat][longg-90]) == NULL)
  {
  get_terrain(lat,longg);
  ptr = myworld[lat][longg-90];
  }

if(ptr!=NULL)
 {
  /* adjust bounds depending on what cell were drawing */
  adjust_bounds(10,cel,&max1x,&min1x,&max1z,&min1z);
  adjust_bounds(100,cel,&max2x,&min2x,&max2z,&min2z);
  adjust_bounds(1200,cel,&max3x,&min3x,&max3z,&min3z);

  /* calculate overlap boundaries */
  maxsub2x = (max2x/10);/*level2 boundary with level1*/
  maxsub2z = (max2z/10);
  minsub2x = (min2x/10);
  minsub2z = (min2z/10);

  maxsub3x = (max3x/12);/*level3 boundary with level2.*/
```

```
  maxsub3z = (max3z/12);
  minsub3x = (min3x/12);
  minsub3z = (min3z/12);

/* adjust so will draw correctly when looking south and west */
/* stops the loop from accessing out of the array */
  if(max1x==10)
    max1x = 9;
  if(max1z==10)
    max1z = 9;
/*************************************************************/

  /* draw level 1 area with all three resolutions */
  for(i=min1x;i<=max1x;i++)
   for(j=min1z;j<=max1z;j++)
   {
   if(!((i>=minsub2x)&&(i<maxsub2x)&&(j>=minsub2z)&&(j<maxsub2z)))
/* draw level 1 square */
     {
     make_polly((i+j)%2,(i*CELL_SIZE1),-(j*CELL_SIZE1),
           ptr->level1[i][j],ptr->level1[i][j+1],
           ptr->level1[i+1][j+1],ptr->level1[i+1][j],
           CELL_SIZE1);
     }
   }/* level 1 area */

/* draw level2 area */
  for(s=(minsub2x*10);s<(maxsub2x*10);s++)
   for(t=(minsub2z*10);t<(maxsub2z*10);t++)
   {
   if(!((s>=minsub3x)&&(s<maxsub3x)&&
     (t>=minsub3z)&&(t<maxsub3z)))
     {
     make_polly( (s+t)%2,(s*CELL_SIZE2), -(t*CELL_SIZE2),
          ptr->level2[s][t],ptr->level2[s][t+1],
          ptr->level2[s+1][t+1],ptr->level2[s+1][t],
          CELL_SIZE2);
/* to cover up diffrences in resolution boundaries
  don't need between level 1 and 2 too far to be noticeable */
     if((s==minsub3x)||(s==maxsub3x))
      {
      draw_skirt(s,-t,ptr->level2[s][t],
            s,-(t+1),ptr->level2[s][t+1],
             CELL_SIZE2);
      }
```

```
      if((t==minsub3z)||(t==maxsub3z))
        {
        draw_skirt(s,-t,ptr->level2[s][t],
              (s+1),-t,ptr->level2[s+1][t],
                CELL_SIZE2);
        }
      }
    }/* level 2 area */

/* draw level3 area */
   for(u=(minsub3x*12);u<(maxsub3x*12);u++)
     for(v=(minsub3z*12);v<(maxsub3z*12);v++)
     {
      make_polly( (u+v)%2,(u*CELL_SIZE3), -(v*CELL_SIZE3),
           ptr->level3[u][v],ptr->level3[u][v+1],
           ptr->level3[u+1][v+1],ptr->level3[u+1][v],
           CELL_SIZE3);
     }/* level 3 area */

  }/* if cell has value */
}
```

APPENDIX G - HIERARCICAL DATA STRUCTURE WITH POINTERS

These are the structure dependent routines used with the multiple array data structure discussed in Chapter five. The procedure *3t.c* is used to preprocess the DMA digital terrain elevation data and place the processed data in a file format that is easily read by the application program. The terrain data is initially read into the application program's structure by the procedure *get_terrain()*. The final structure dependent procedure is *draw_cell()*. It is used to actually access the data structure and draw the visible terrain.

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* FnName:      3t.c
* Author:      FRANK HARRIS
* Date:        mar 88
* Purpose:     make 3 tier data structure file for
               display terrain. Uses a hierarchical structure
               with pointers to store the terrain.
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#define maxelev 10000/* in yards higher the everest */
#define FALSE 0
#define TRUE  1
#define buf_size 28672
char *BUF,*BUFO;
char *malloc();
short   terrain[1201][1201];

typedef struct{
        short      data[13][13];
      }level3_rec;
typedef level3_rec *ptr3;

typedef struct{
        ptr3       level3ptr[11][11];
        short      level3val[11][11];
        short      all_zero;
      }level2_rec;
typedef level2_rec *ptr2;

typedef struct{
        ptr2       level2ptr[11][11];
        short      level2val[11][11];
```

```
        short      all_zero;
      }level1_rec;
typedef level1_rec *ptr1;

typedef ptr1 octant[90][90];
octant myworld;
ptr1  do_level1();
ptr2  do_level2();
ptr3  do_level3();

main(argc,argv)
int argc;
char *argv[];
{
      int  row,j,i,n,nbyte;
      char *name;
      char c[2];
      short *P,lat,log;
      FILE       *fdi,*fdo;
     /* open input file */
      if((fdi=fopen(argv[1],"r")) < 0)
      {
            printf("cannot open file0);
            exit();
      }

/* set up input buffer */
      if ((BUF=malloc((buf_size )+1)) == NULL)
      {
       fprintf(stderr,"out of memory0);
       exit(1);
      }
      setvbuf(fdi,BUF,_IOFBF,buf_size );

/* read 1 colume of data at a time and put in an array
  the first index is the colume number "longitude"
  the second is for the row or latitude
  index [0][0] is the lower left corner of the cell*/
      for (i=0;i<=1200;i++)
      {
       P = &terrain[i][0];
       nbyte = fread(P,2,1201,fdi);
      }

/* close input file no longer needed */
```

```
free(BUF);
fclose(fdi);

/* make output file name */
      name = argv[1];
      strcat(name,".3t");

      /* open output file */
      if((fdo=fopen(name,"w")) < 0)
      {
            printf("cannot open output file0);
            exit();
      }

/* setup output buffer */
      if ((BUFO=malloc((buf_size )+1)) == NULL)
      {
       fprintf(stderr," out of memory");
       exit(1);
      }
      setvbuf(fdo,BUFO,_IOFBF,buf_size );

/* convert lat long in file name to short int values */
      if (name_conv(&lat,&log,argv[1]))
      {
       printf(" filename not propper LAT/LONG");
       exit(1);
      }


/* start the file with lat long of lower left corner */
      nbyte=fwrite(&lat,2,1,fdo);
      nbyte=fwrite(&log,2,1,fdo);

/* make structure put pointer in world structure */
      myworld[lat][log-90] = do_level1();

/* put overlap data in each cell */
      insert_overlap(myworld[lat][log-90]);
      out_terrain(myworld[lat][log-90]);

/* writeout the structure to reuseable file */
      writeit(fdo,myworld[lat][log-90]);

/* clean up */
```

```
        fclose (fdo);
        free(BUFO);
}
/*************************************************************/

out_terrain(pt)
ptr1 pt;
{
int s,t,i,j;
ptr2 pt2;
  printf(" level1 ");
for(i=10;i>=0;i--)
{
  printf("  ");
  for(j=0;j<11;j++)
    {
    printf("%5d ",pt->level2val[j][i]);
    }
}
printf("  ");
for (i=0;i<11;i++)
  for(j=0;j<11;j++)
   {
  pt2 = pt->level2ptr[i][j];
  if (pt2 == NULL)
     printf(" null level2 i %d j %d ",i,j);
  else
    {
     printf(" level2 i %d j %d ",i,j);
    for(s=10;s>=0;s--)
    {
     printf("  ");
     for(t=0;t<11;t++)
       {
       printf("%5d ",pt2->level3val[t][s]);
       }
    printf("  ");
    }
}
  }
}
/*************************************************************/
insert_overlap(ptr)
ptr1  ptr;
{
```

```
if (ptr != NULL)
  {
int i,j,s,t;
ptr2 l2,top2,rt2;
ptr3 l3,top3,rt3;

/* put in overlap  for level1*/
  for (j=0;j<10;j++)
    ptr->level2val[10][j] = terrain[1200][j*120];
  for(i=0;i<11;i++)
    ptr->level2val[i][10] = terrain[i*120][1200];
  ptr->level2val[10][10] = terrain[1200][1200];


/* overlap for level2 */
for (i=0;i<10;i++)
  for(j=0;j<10;j++)
    {
    l2 = ptr->level2ptr[i][j];
    if (l2 != NULL)
      {
      top2 = ptr->level2ptr[i][j+1];
      if (top2 != NULL)
        for (s=0;s<10;s++)
          l2->level3val[s][10] = top2->level3val[s][0];
      else
        for (s=0;s<10;s++)
          l2->level3val[s][10] = 0;
      rt2 = ptr->level2ptr[i+1][j];
      if (rt2 != NULL)
        for (s=0;s<10;s++)
          l2->level3val[10][s] = rt2->level3val[0][s];
      else
        for (s=0;s<10;s++)
          l2->level3val[s][10] = 0;
      rt2 = ptr->level2ptr[i+1][j+1];
      if (rt2 != NULL)
          l2->level3val[10][10] = rt2->level3val[0][0];
      else
          l2->level3val[10][10] = 0;
      }
    }
  for (i=0;i<9;i++)
    {
    l2 = ptr->level2ptr[i][9];
```

```
    if (l2 != NULL)
      {
      for (j=0;j<10;j++)
        {
        /* for outside border */
        l2->level3val[j][10] = terrain[i*120+j*12][1200];

        /* for right side with adjacent subcell */
        rt2 = ptr->level2ptr[i+1][9];
        if(rt2 != NULL)
         l2->level3val[10][j] = rt2->level3val[0][j];
        else
         l2->level3val[10][j] = 0;
        }
      l2->level3val[10][10] = terrain[i*120+120][1200];
      }
    l2 = ptr->level2ptr[9][i];
    if (l2 != NULL)
      for (j=0;j<10;j++)
        {
        /* for outside border */
        l2->level3val[10][j] = terrain[1200][i*120+j*12];

        /* for top side with adjacent subcell */
        rt2 = ptr->level2ptr[9][i+1];
        if(rt2 != NULL)
         l2->level3val[j][10] = rt2->level3val[j][0];
        else
         l2->level3val[j][10] = 0;
        }
   }
  l2 = ptr->level2ptr[9][9];
  if (l2 != NULL)
    l2->level3val[10][10] = terrain[1200][1200];
 }
}
/*************************************************************/
writeit(fdo,l1ptr)
FILE *fdo;
ptr1  l1ptr;
{
ptr2  l2ptr;
ptr3  l3ptr;
short allzero,edge;
short value;
```

```
int i,j,s,t,u,v;
allzero = -1;
edge = -2;

if (l1ptr->all_zero) /* entire cell is 0 all ocean */
  {
  fwrite(&allzero,2,1,fdo);/* write a -1 for allzeros */
  }
else /* need to check cell */
  {
  value = 1;/* 1 means data -1 means allzero */
  fwrite(&value,2,1,fdo);
  for (i=0;i<11;i++)/* loop through level1 */
  for (j=0;j<11;j++)
    {
    l2ptr = l1ptr->level2ptr[i][j];
     if((i==10)||(j==10))
      {
printf(" 1 edge i %d j %d ",i,j);
      fwrite(&edge,2,1,fdo);
      fwrite(&(l1ptr->level2val[i][j]),2,1,fdo);
      }
    else if(l2ptr==NULL)/* subcell is empty, all zeros */
      {
      fwrite(&allzero,2,1,fdo);
      }
    else/* subcell has value, not all zeros */
      {
      fwrite(&(l1ptr->level2val[i][j]),2,1,fdo);
      for(s=0;s<11;s++)/* loop through level2 */
      for(t=0;t<11;t++)
        {
        l3ptr = l2ptr->level3ptr[s][t];
        if((s==10)||(t==10))
          {
printf(" 2 edge i %d j %d s %d t %d ",i,j,s,t);
          fwrite(&edge,2,1,fdo);
          fwrite(&(l2ptr->level3val[s][t]),2,1,fdo);
          }
        else if(l3ptr==NULL)/* subsubcell is empty, all zeros */
          {
          fwrite(&allzero,2,1,fdo);
          }
        else/* subsubcell has value, not all zeros */
          {
```

```
        fwrite(&(l2ptr->level3val[s][t]),2,1,fdo);
          {
          fwrite(l3ptr->data,2,169,fdo);
          }
        }
       }
      }
    }
  }
}

/*********************************************************/
ptr1  do_level1()
{
ptr1 level;
ptr2 ptr;
int       i,j;
short     allzero;

/* allocate structure */
level = (ptr1)malloc(sizeof(level1_rec));
/* make the formattred output file */
/* loop through the level 2 data */
for(i=0;i<10;i++)
  for(j=0;j<10;j++)
  {
   ptr = do_level2(i,j,&(level->level2val[i][j]));
   if(ptr != NULL)
     allzero = FALSE;
   level->level2ptr[i][j]=ptr;
  }

level->all_zero = allzero;
return(level);
}
 /*********************************************************/
ptr2 do_level2(i,j,value)
int    i,j;
short   *value;
{
ptr2  level;
ptr3  ptr;
int       s,t;
short     min;
short     allzero;
```

```
/* allocate structure */
/* allocate level2 data record */
min = maxelev;
level = (ptr2)malloc(sizeof(level2_rec));
allzero =TRUE;
/* loop through getting level 3 data */
    for(s=0;s<10;s++)
      for(t=0;t<10;t++)
       {
        ptr = do_level3(i,j,s,t,&(level->level3val[s][t]));
        if(ptr != NULL)
             allzero = FALSE;
        level->level3ptr[s][t]=ptr;
        if (level->level3val[s][t] < min)
          min = level->level3val[s][t];
       }
if (allzero)
  {
/* all zero don't need the data  */
  *value = 0;
  free(level);
  return(NULL);
  }
else
  {
/* pass back pointer to the record and min value of area */
  level->all_zero = allzero;
  *value = min;
  return(level);
  }
}
/*********************************************************/
ptr3 do_level3(i,j,s,t,value)
int i,j,s,t;
short    *value;
{
int u,v;
level3_rec *level;
short min,temp,allzero;

min = maxelev;
/* get data record for 3rd level */
level = (ptr3)malloc(sizeof(level3_rec));
allzero = TRUE;/* initialize */
/* loop through terrain */
```

```
for (u=0;u<13;u++)
  for (v=0;v<13;v++)
  {
    temp = terrain[i*120+s*12+u][j*120+t*12+v];
    level->data[u][v]= temp;
    if (temp!=0)
      allzero = FALSE;
    if (temp<min)
      min = temp;
  }
if (allzero)
  {
/* all zero don't need to keep the points around */
  *value = 0;
  free(level);
  return(NULL);
  }
else
/* return the pointer to the record and the min value of the points */
  {
  *value = min;
  return(level);
  }
}

/*********************************************************/
int name_conv(lat,log,name)
short *lat,*log;
char  *name;
{
short temp;
/* assume all lats and longs are north and east respectively */

temp=(short)name[0]-48;
if ((temp<0)||(temp>9))
   return(-1);
*lat = temp * 10;
temp=(short)name[1]-48;
if ((temp<0)||(temp>9))
   return(-1);
*lat = *lat + temp;

temp=(short)name[3]-48;
if ((temp<0)||(temp>9))
   return(-1);
```

```
*log = temp * 100;
temp=(short)name[4]-48;
if ((temp<0)||(temp>9))
   return(-1);
*log = ((temp*10) + *log);
temp=(short)name[5]-48;
if ((temp<0)||(temp>9))
   return(-1);
*log = (temp + *log);
return(0);
}
```

```
#include <string.h>
#include <sys/param.h>
#include <sys/types.h>
#include <sys/times.h>
#include "gl.h"
#include "device.h"
#include "constants.h"
#include "typedef.h"
#include "stdio.h"


#define maxelev 10000/* in yards higher the everest */
#define buf_size 28672/* for input buffer */

char *malloc();
char *BUF;

extern octant myworld;
ptr1  make_level1();
ptr2  make_level2();
ptr3  make_level3();
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* FnName:      get_terrain.c
* Author:      FRANK HARRIS
* Date:        feb 88
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

get_terrain(lat,longg)
short  lat,longg;
{
int  row,j,i,n,nbyte;
char name[10],filename[100];
short *P,buf[512];
FILE       *fdi;


name_conv(lat,longg,name);
strcpy(filename,"/usr/work/fharris/3t_data/");
strcat(filename,name);
strcat(filename,".3t");

/* open input file */
if((fdi=fopen(filename,"r")) < 0)
{
 printf("cannot open file0);
```

```
 myworld[lat][longg-90] = NULL;
}
else
{
 /* set up input buffer */
 if ((BUF=malloc((buf_size )+1)) == NULL)
 {
  fprintf(stderr,"out of memory0);
  myworld[lat][longg-90] = NULL;
 }
 setvbuf(fdi,BUF,_IOFBF,buf_size );

 printf(" before first read ");

 /* get lat long first 2 items in file */
 fread(buf,2,2,fdi);

 /* make structure put pointer in world structure */
 myworld[lat][longg-90] = make_level1(fdi);

 /* clean up */
 fclose (fdi);
 free(BUF);
}
}

/*********************************************************/
ptr1  make_level1(fdi)
FILE *fdi;
{
ptr1 level;
ptr2 ptr;
int      i,j;
short    buf[512],allzero;

/* allocate structure */
level = (ptr1)malloc(sizeof(level1_rec));
/* loop through the level 2 data */
fread(buf,2,1,fdi);
if(buf[0]==-1)
  level->all_zero = TRUE;
else
{
 level->all_zero = FALSE;
 for(i=0;i<11;i++)
```

```
    for(j=0;j<11;j++)
    {
    level->level2ptr[i][j] = make_level2(fdi,&(level->level2val[i][j]));
    }
}
return(level);
}
/*********************************************************/
ptr2 make_level2(fdi,value)
FILE    *fdi;
short   *value;
{
ptr2  level;
ptr3  ptr;
int       s,t;
short     min;
short     allzero;

fread(value,2,1,fdi);
if (*value == -1)
   {
   *value = 0;
   return(NULL);
   }
else if (*value == -2)
   {
   fread(value,2,1,fdi);
   return(NULL);
   }
else
   {
   /* allocate level2 data record */
   level = (ptr2)malloc(sizeof(level2_rec));
   /* loop through getting level 3 data */
   for(s=0;s<11;s++)
     for(t=0;t<11;t++)
     {
     level->level3ptr[s][t]= make_level3(fdi,&(level->level3val[s][t]));
     }
   }
return(level);
}
/*********************************************************/
ptr3 make_level3(fdi,value)
FILE     *fdi;
```

```
short    *value;
{
int u,v;
level3_rec *level;

fread(value,2,1,fdi);
if (*value == -1)
   {
   *value = 0;
   return(NULL);
   }
else if (*value == -2)
   {
   fread(value,2,1,fdi);
   return(NULL);
   }
else
   {
/* get data record for 3rd level */
   level = (ptr3)malloc(sizeof(level3_rec));
   fread(level->data,2,169,fdi);
   return(level);
   }
}

/*********************************************************/
/* pass in lat/long will return filename */

name_conv(lat,longg,name)
short lat,longg;
char  name[10];
{
/* assume all lats and longs are north and east respectively */
char    c[2];

c[1]= '  ';
name[1]='  ';
name[0] = (char)((lat/10)+48);
c[0] = (char)((lat%10)+48);
strcat(name,c);
c[0] = 'N';
strcat(name,c);
c[0] = (char)((longg/100)+48);
strcat(name,c);
longg=longg%100;
```

```
c[0] = (char)((longg/10)+48);
strcat(name,c);
c[0] = (char)((longg%10)+48);
strcat(name,c);
c[0] = 'E';
strcat(name,c);
printf(" %s",name);
}
```

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* FnName:      Draw_cell
* Author:      FRANK HARRIS
* Purpose:     Draws on e cell of terrain based on the
               input parameters.s version uses the
               pointer data structure
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

draw_cell(cell,lat,longg,max1x,min1x,max1z,min1z,
                  max2x,min2x,max2z,min2z,
                  max3x,min3x,max3z,min3z)
short cell,lat,longg;
int   max1x,min1x,max1z,min1z;
int   max2x,min2x,max2z,min2z;
int   max3x,min3x,max3z,min3z;
{
int maxsub2x,maxsub2z,minsub2x,minsub2z;/* overlap bounds */
int maxsub3x,maxsub3z,minsub3x,minsub3z;
int stx1,stz1,spx1,spz1;/* start stop points for dif resolutions */
int stx2,stz2,spx2,spz2;
int stx3,stz3,spx3,spz3;
int i,j,s,t,u,v,count;
ptr1 ptr;
ptr2 cptr;
ptr3 sptr;

/* get cell record */
if ((ptr = myworld[lat][longg-90]) == NULL)
  {
  get_terrain(lat,longg);
  ptr = myworld[lat][longg-90];
  }

if(ptr!=NULL)
{
  /* adjust bounds depending on what cell were drawing */
  adjust_bounds(10,cell,&max1x,&min1x,&max1z,&min1z);
  adjust_bounds(100,cell,&max2x,&min2x,&max2z,&min2z);
  adjust_bounds(1200,cell,&max3x,&min3x,&max3z,&min3z);

  /* calculate overlap boundaries */
  maxsub2x = max2x/10;/*level2 boundary with level1. in level1 coord */
  maxsub2z = max2z/10;
  minsub2x = min2x/10;
  minsub2z = min2z/10;
```

```
  maxsub3x = max3x/12;/*level3 boundary with level2. in level2 coord */
  maxsub3z = max3z/12;
  minsub3x = min3x/12;
  minsub3z = min3z/12;

/* adjust so will draw correctly when looking south and west */
/* stops the loop from accessing out of the array */
  if(max1x==10)
    max1x = 9;
  if(max1z==10)
    max1z = 9;

count = 0;
  /* draw level 1 area with all three resolutions */
  for(i=min1x;i<=max1x;i++)
   for(j=min1z;j<=max1z;j++)
   {
   if((i>=minsub2x)&&(i<=maxsub2x)&&(j>=minsub2z)&&(j<=maxsub2z))
/* in level 2 area,draw level2 resolution */
    {
    cptr = ptr->level2ptr[i][j];
    if(cptr!=NULL)
     {
     for(s=0;s<10;s++)
      for(t=0;t<10;t++)
      {
      if((((i*10)+s)>=minsub3x)&&(((i*10)+s)<=maxsub3x)&&
        (((j*10)+t)>=minsub3z)&&(((j*10)+t)<=maxsub3z))
      /* check if overlap level3 data draw level 3 resolution*/
       {
       sptr = cptr->level3ptr[s][t];
       if(sptr!=NULL)
        {
        for(u=0;u<12;u++)
         for(v=0;v<12;v++)
         {
        make_polly( (u+v)%2,
              ((i*CELL_SIZE1)+(s*CELL_SIZE2)+(u*CELL_SIZE3)),
             -((j*CELL_SIZE1)+(t*CELL_SIZE2)+(v*CELL_SIZE3)),
           sptr->data[u][v],sptr->data[u][v+1],
           sptr->data[u+1][v+1],sptr->data[u+1][v],
           CELL_SIZE3);
         }
        }
       }
```

```
        else
        /* draw level 2 resolution */
          {

          make_polly( (s+t)%2,(i*CELL_SIZE1+s*CELL_SIZE2),
                -(j*CELL_SIZE1+t*CELL_SIZE2),
              cptr->level3val[s][t],cptr->level3val[s][t+1],
              cptr->level3val[s+1][t+1],cptr->level3val[s+1][t],
              CELL_SIZE2);
          }
/* to cover up diffrences in resolution boundaries
  don't need between level 1 and 2 too far to be noticeable */
        if((((i*10)+s)==minsub3x)||(((i*10)+s)==(maxsub3x)))
          {
          draw_skirt(((i*10)+s),-((j*10)+t),cptr->level3val[s][t],
                ((i*10)+s),-((j*10)+t+1),cptr->level3val[s][t+1],
                  CELL_SIZE2);
          }
        if((((j*10)+t)==minsub3z)||(((j*10)+t)==(maxsub3z)))
          {
          draw_skirt(((i*10)+s),-((j*10)+t),cptr->level3val[s][t],
                ((i*10)+s+1),-((j*10)+t),cptr->level3val[s+1][t],
                  CELL_SIZE2);
          }
        }
       }
     }
    else
    /* draw level 1 resolution */
      {
count = count + 1;
     make_polly((i+j)%2,(i*CELL_SIZE1),-(j*CELL_SIZE1),
          ptr->level2val[i][j],ptr->level2val[i][j+1],
          ptr->level2val[i+1][j+1],ptr->level2val[i+1][j],
          CELL_SIZE1);
     }
    }/* level 1 area */
}/* if cell has value */
printf(" polly count %d ",count);
}
```

LIST OF REFERENCES

1. Sanders, Mark S. and McCormick, Ernest J., *Human Factors in Engineering and Design,* sixth edition, McGraw-Hill, 1987.
2. Adams, Rodney M., *A Software Architecture for a Commanders Display System,* Master's Thesis, Naval Postgraduate School, Monterey, California, April 1987.
3. Smith, Douglas and Streyle, Dale, Master's Thesis, Naval Postgraduate School, Monterey, California, July 1987.
4. Shufeldt, Henry H., *The Calculator Afloat,* Naval Institute Press, Annapolis, Maryland, 1980.
5. Griggs, Laurence W., *An Interactive Computer Graphics Network Monitor for a tactical Communications Network,* Master's Thesis, Naval Postgraduate School, Monterey, California, August 1987.
6. Zyda, Michael J., *Inexpensive, Real-Time, Interactive, three-Dimensional, Visual Simulators,* Lecture Notes for a talk given at Princeton University, Princeton, New Jersey, May 1988.
7. Oliver, Michael and Stahl, David, *Interactive, Networked, Moving Platform Simulators,* Master's Thesis, Naval Postgraduate School, Monterey, California, Febuary 1988.
8. *IRIS User's Guide,* Version 3.0, v. 1, Silicon Graphics, Inc., Mountainview, California, 1987.
9. *Using MEX, The IRIS Window Manager,* v. 1, Silicon Graphics, Inc., Mountainview, California, 1987.
10. Office of the Federal Register, National Archives and Records Administration, *The United States Government Manual,* Government Printing Office, Washington, D.C., June 1987.
11. Bowditch, Nathaniel, *American Practical Navigator,* DMA PUB no. 9, v. 2, Defense Mapping Agency Hydrographic/Topographic Center, Washington, D.C., 1981.
12. *Product specifications for Digital Landmass System Database,* Defence Mapping Agency, Washington, D.C., 1983.
13. Rogers, David F., *Procedural Elements for Computer Graphics,* McGraw-Hill, 1985.

Distribution List for Dr. Michael J. Zyda

Defense Technical Information Center,
Cameron Station,
Alexandria, VA 22314 — 2 copies

Library, Code 0142
Naval Postgraduate School,
Monterey, CA 93943 — 2 copies

Center for Naval Analyses,
4401 Ford Avenue
Alexandria, VA 22302-0268 — 1 copy

Director of Research Administration,
Code 012,
Naval Postgraduate School,
Monterey, CA 93943 — 1 copy

Dr. Michael J. Zyda
Naval Postgraduate School,
Code 52, Dept. of Computer Science
Monterey, California 93943-5100 — 200 copies

Mr. Bill West,
HQ, USACDEC,
Attention: ATEC-D,
Fort Ord, California 93941 — 1 copy

John Maynard,
Naval Ocean Systems Center,
Code 402,
San Diego, California 92152 — 1 copy

Duane Gomez,
Naval Ocean Systems Center,
Code 433,
San Diego, California 92152 — 1 copy

James R. Louder,
Naval Underwater Systems Center,
Combat Control Systems Department,
Building 1171/1,
Newport, Rhode Island 02841 — 1 copy